Chapter 1

INTRODUCTION

Software Configuration Management (SCM) is the discipline which deals with the procedures or methods that involve in controlling the evolution of complex software systems. Software Product Line (SPL) is a family of products that increases reusability, reduces development effort besides increasing productivity. It is the paradigm shift moving from individual products to product lines. The new paradigm facilitates identifying generic solutions, building related products, reusing components to build common solutions. Lessons learned from the SPL helped software industry to adapt more useful industry standards in product line development and configuration. Component-based development is reinforced in SPL. The components which are prefabricated with standards can avoid reinventing the wheel beside positively affecting the timeline and budgets of software development. The components and services of SPL are structured using an architecture that satisfies application domain or business goals. The existing solutions found in the literature focused on different aspects of SPL and there was no comprehensive framework found to take care of SPL configuration management and product derivation. The following sections throw light on basic concepts, motivation, objectives and scope of the proposed work along with organization of thesis.

1.1 IMPORTANCE OF SOFTWARE ENGINEERING

Software development was started long back with trial and error basis. As the complexity in software increased, a systematic approach was needed to develop quality software. Thus a new branch of computer science came into existence. It is known as software engineering which provides many process models. Software process model is a methodology that guides developers to build good software. Good software is the software that can withstand to various runtime situations besides meeting all functional requirements of the client. As the software is increasingly used in every field it became imperative to have software engineering principles in place.
for building quality software. Due to the invention of distributed technologies, software development became very complex and needs engineering for handling ever-growing complexity.

With modern software process models like Agile methodology, it is possible to build software in time and improve customer satisfaction [1]. With such engineering discipline, there will be many disadvantages. They include difficulty in meeting deadlines for various milestones, cost overruns, errors in software even after testing, high cost in software maintenance, and issues measuring the process of software development. Building software is different from manufacturing other products. Software development approach needs to understand the characteristics of software. They include intangibility, longevity, and needs regular maintenance. In order to address all these requirements, software process models are needed for building high-quality software. The usage of process model depends on the kind of project being developed. Nevertheless, certain things are common in all process models. They include software specification, design, implementation, testing, and maintenance. These are part of software development life cycle. [2] Now the software development is driven by standards such CMMI (Capability Maturity Model Integration) which guide software development firms to have culture and discipline to produce best quality software.

Of late software, quality is given high importance. As systems are never developed and they are being developed, it is a continuous process which makes the quality products. When software quality is not maintained with continuous improvement it becomes obsolete. SPL is not an exception to this. SPL when monitored for quality product derivation and configuration management it improves the longevity of product family. CMMI can be integrated into the SPL development process in order to have the culture to produce quality product lines.

1.2 IMPORTANCE OF CONFIGURATION MANAGEMENT

Configuration Management (CM) is an essential activity that is used to identify items in a system and configure them, control their release and change
management. As software became unduly complex, it is inevitable to have a mechanism to keep track of different components or artifacts and their versions. Failing to do so results in chaos and that is the reason CM is important [3]. As software development life cycle involves plenty of change requests from the client, it is indispensable to have software to manage the production of other software. Such software is known as configuration management software. As there are many stakeholders who need to know the status of every component in software and different improvements over a period of time while retaining old versions, CM has assumed unprecedented significance. As the development team needs to have coordinated effort because the change in an artifact shall have an effect on the related artifacts, they need to have a common configuration system accessible and usable.

Configuration management is required by hardware projects as well. However, there are subtle differences between them as the nature of software engineering projects is different. The software configuration management system is, therefore, a system that is used to manage the evolution of software products right from the inception to the entire life of the software. Even at the time of initial development SCM plays a vital role. A software product is a collection of related programs, procedures, related documentation and data which are to be delivered to users. SCM also takes care of supporting material which is not actually part of software [4]. Therefore SCM is a set of activities carried out during software engineering process in order to determine and identify artifacts, ensure needed documentation, controlled change management, accurate usage of different versions, and ascertain the status of artifacts at any given point of time.

Activities in SCM are configuration identification, change management, configuration status accounting, release process, reviews, and audits. [5] As the software engineering is an iterative process SCM is required to track the state of artifacts and their enhancements from time to time throughout the life cycle. Therefore it is a challenging task to have configuration management in order to produce high quality products. SCM has a common point where different things that span life cycle are integrated. It is the framework that provides an interface for different activities and related changes being tracked. As software products are
developed by different groups, SCM needs to have flexibility in such a way that it is common for all groups. SCM also helps in tracking software artifacts along with their supporting hardware and coordinates product evolution from time to time. Visible management is possible with SCM [6].

1.3 ADVENT OF PRODUCT LINES

Software is the heart of modern systems in all walks of life. It is rendering services with unprecedented granularity and scale. Naturally, software has been growing in complexity to fulfill ever-increasing needs of users. As customers are expecting more features and convenience, software needs to support different hardware, different constraints and different market segments. This is one side of the coin. On the other side of the coin, the development companies are under constant pressure to produce software with high quality in very less time due to cut-throat competition in the real world [7]. As it is required for software firms to remain competitive in the market place, they need to think a way out. The solution, fortunately, came in the form of reusing software components. Component technology is adapted from hardware in order to produce interoperable and reusable software components for high productivity and lessen the time to market. Reusability features of software components helped to increase productivity besides reducing cost and effort. This has its impact on the programming languages, architectures and development tools [8].

The process for development of a single product is well known. However, when multiple products are needed with more common features and some variability development complexity increases. Thus complexity in configuration management increases. Stated differently many products share much of the functionality and the complexity is increased by an order of magnitude when compared with a single product. To overcome this problem and to take the reuse of software components to next level, Software Product Line (SPL) came into existence. When compared to single product development it is a paradigm shift to adapt SPL in order to alleviate cost, reduce time to market, exploit large scale reuse, besides supporting multiple market segments. There are some domains in the real world where software products
share many common features like [9] support hardware, standards, features and algorithms. As a matter of fact, product line concept is well known in manufacturing companies such as Boeing and McDonald to mention few. Car manufacturing is the best example of a product line. Cars of different models share certain commonalities to support the Product Line. The key in SPL is to reuse the existing artifacts which are very costly when built from scratch.

There is another aspect in software development industry. It is the usage of modern process model known as the agile model. This model improved the quality of software development. Agility refers to doing things quickly and delivering software on time for increasing the time to market feature. When conventional software development methods are used it is not easy to accommodate new requirements. With agile process model, it is possible to accommodate new requirements with ease. The model treats the requirements as user stories and supports iterative process in the form of sprints. As each sprint is completed in less number of days and delivered to the customer for feedback, the [9] Agile process model increases customer satisfaction significantly. Moreover, it allows the client and development team to work closely which results in high success rate. According to agile software development manifesto, the focus of agile model includes interactions over tools and processes, software with comprehensive documentation, closely working with the customer in collaboration, and the ability to respond to changes. There is a good relationship between all stakeholders including developers of the project. Deliveries are given to the client in less time so as to help the client to give feedback. The feedback, in turn, helps the team to improve the quality of software. Even when there are technical difficulties, that is visible to the client so that the client also can ascertain the situation and cooperate by giving flexibility in time.

Agile processes are incremental, cooperative, straight and with adaptability. There are many agile models known as extreme programming (EP), scrum and feature driven development. The basic principles of all the agile models include embracing change as change is inevitable, rapid feedback, assuming simplicity, incremental change and producing quality work. There are certain practices in agile models. They include product backlog, effort estimation, and sprint. When agile process model is
applied to SPL, it becomes agility product line. According to Noor et al. [1] both SPL engineering and agile methods proved to increase customer satisfaction besides reducing time to market. The key characteristics they found in Agile and SPL combination are highly iterative development, lean and there is a strong emphasis on the involvement of stakeholders. The product line planning and development when coupled with agile methods can have tangible benefits in SPL development. When teams involved in agile process modeling and the product line engineering have a shared common goal, it renders significant quality in production. When agility is part of SPL development, it naturally provides the required approach into the framework of product line development.

Product line needs evolution support. More information on the new product derivation is provided in the ensuing section. The process evolution in product line refers to identifying changes and incorporating in the development process. When there is a change request, it has its course of action to be followed. When the change is related to a specific product, that product is affected besides many global artifacts. The evolution is not confined to products only. In fact they encompass to architecture evolution, product line evolution and product evolution. The end result is new variant or new product which again forces the configuration management of SPL to keep track of these changes. Therefore there is close relationship between product and produces evolution and configuration management which is covered in the next section.

The change management process which is part of SPL includes many activities such as architecture change management, product line change management, component change management, and product change management. The change processes include identification of change, change impact, change propagation and validation of the change. The changes that are dealt by the SPL include anticipative changes, corrective changes, perfective changes, [10] and adaptive changes. In order to have consistency, there needs to change propagation. Change propagation is the process of ensuring that the change introduced in the SPL at any level should have the consistency across all artifacts and products. Stated differently, change propagation is an essential activity to ensure that the SPL remains up to date with respect to the
change introduced. An SPL has product line architecture, product line, product and artifacts [11]. There are inter and intra-relationships. Again the architecture is at different levels of SPLE. They are base product architecture, domain architecture, product line architecture and product architecture.

The advent of product lines in the software industry has brought significant change in the thinking process of development teams and organizations. Many hypotheses were involved in the thinking process. They include the relationship between software product line and domain engineering, the relationship between the performance of product line and requirements engineering, the role of managing commonality in product line, the relationship between product line and variability management, the relationship between performance of product line and architecture evaluation, and the relationship between artifacts management and the performance of product line. Towards the hypotheses, many architectural factors are involved. They include architecture [12] artifacts management, architecture evaluation, variability management, commonality management, requirements modeling, and domain engineering. When all these activities are integrated into a cohesive SPL framework, it results in SPL performance.

1.4 IMPORTANCE OF CONFIGURATION MANAGEMENT AND QUALITY PRODUCT DERIVATION IN SOFTWARE PRODUCT LINES

SPL has many artifacts which are reusable and are called assets. The code implemented in SPL to have a common feature is shared by many products. SCM is already described earlier in this chapter. However, SCM for SPL needs to be understood. SPL poses challenges to SCM when compared with that of single product development. When there is a single product, its evolution is in time dimension only. It is treated as a single evolving entity. In the case of SPL, multiple products evolve independently and many components are shared by them. Products and artifacts have their own line of development in SPL. Artifacts are evolved in time dimension while products are evolved in space dimension [13]. Space refers to the actual space in the product line. Developers of artifacts and developers of products may be different. The
sharing of artifacts across the product line forms a network of products with component dependency.

A change in an artifact can affect many other components and products in the SPL. Therefore, SCM needs to handle this ripple effect effectively. When an interface of a component is changed, it causes changes in all implementing components and products. This forces the SCM of SPL to handle changes correctly. Such dependencies are found in code, database and documentation. The conventional approach to handle this is to use branching capability of traditional SCM. The problem here is that once a branch is created, the communication is lost with other branches and assets. With such SCM product evolution results in incompatibilities. This causes reimplementation of certain products or assets. The solution to this is to have special SCM approach for product line.

Under configuration management, there is no direct intra-dependency in artifacts. Interestingly a product in use is not under configuration management. Therefore the changes made to custom and core assets cannot be reflected in the product in use. However, they are available for new products in future. In the same fashion, the changes made to the product in use cannot be reflected in the core and custom artifacts. When more changes are made to the product in use, it is more deviated from the product line in question. Therefore, it is very important understanding in software product line that SPL configuration management ignores inter-relationships among products. However, it is tedious to track origination of a product from product line without proper configuration management. From the above discussion, it is clear that the artifacts that come under configuration management are core assets, custom assets and production while the products in use outside the purview of configuration management. When corrective changes are made to a product in use, it is essential to ensure that the changes are reflected in corresponding core and custom assets. Thus it makes sense when assets are changed and products are updated.

There is the concept of evolution-based configuration management. According to this model, a product in use has its corresponding product instance that comes under the purview of configuration management in the production domain. The
changes made to core and custom assets can propagate to corresponding products in use through specific means such as update and release. It is very important to cover core part of products in the configuration management. This capability is provided by evolution-based configuration management. Configuration management can help improve product evolution. In fact, the evolution-based configuration management can help in preventing product deviation or product line decay. This is possible with proper configuration management so as to improve the quality of products.

There are some configuration management issues involved in this model. They include version management, branch management, concurrency management, asset creation, asset evolution, asset branch management, asset baseline management, and variant management. These are various configuration management activities that can cause issues when not handled properly. These activities are related to certain artifacts. For instance version management can deal with components, assets, core instance, and product instance. Concurrency management is applicable to component, assets and product instances. Evolution management is also applicable to all aforementioned artifacts. Branch management is applicable to only components and assets. Baseline management is applicable to only assets while the variant management is applicable to core instance and product instance.

The configuration management needs to deal with different kinds of changes in the product line. They are corrective changes and enhancement changes. With respect to corrective changes, core assets are input assets to produce a product in a product line. With respect to enhancement changes, the inputs are not from core and custom assets but from request and request feedback are considered in the core part of the product. From the discussion, it is well understood that managing core instance with configuration management has less workload when compared with that of product instance. Therefore, it is best used when products have the major part as core part as various products can share a common core instance. When core part is very small, the benefits of the model are less.

Product derivation is purely customer-driven. In other words, new products are derived based on the genuine customer needs. Therefore, it is essential to derive high quality products in order to satisfy the target customers. As the target customers are
also stakeholders of the product line in one way or the other, their needs are to be fulfilled with acceptable quality. Thus, the product derivation assumed vital importance in SPL. The software development firms and especially product line developers in the software industry face severe competition and they have pressure to bring about customer satisfaction and retain customers. In order to continue to demand customer loyalty and attract new customers, it is required to derive high quality products that cross customer expectations.

There is a relationship between product derivation and configuration management. When the configuration is managed consistently it is possible to have high quality product derivation. The product derivation efficiency is somehow linked to configuration management. For this reason, this thesis focuses on proposing a framework that can support effective configuration management and quality product derivation. The product derivation can depend on the genuine customer needs and the quality requirements. Besides supporting product derivation, the proposed framework supports an algorithm that produces products with different combinations of artifacts. And finally it produces a high quality product.

1.5 DESIGN PATTERNS AND ONTOLOGY

Design pattern is a blueprint or model solution to a design problem that occurs frequently. Alexander Christopher, a construction engineer, started documenting design patterns in the construction industry in the 1970s. This helped construction engineers to reuse designs to solve different problems they face. This concept is adapted to software engineering. In software development also design patterns are recognized as reusable solutions [16] that can help productivity besides reducing cost and time overruns in software development. There are many design patterns related to object oriented programming. Each pattern deals with specific design problem in software development arena. A book written by GOF [17] categorizes design patterns into three categories creational design patterns, structural design patterns, and behavioral design patterns. The first category design patterns take care of different methods to create groups of objects. The second category has methods to organize objects and classes into larger structures while the third
category deals with assigning responsibilities to object and collaboration among objects.

Each design pattern may comprise of a set of classes. In fact, the design patterns are elements reusable software with a name for a pattern, a description on how to apply to a problem, a solution to describe the pattern, its components, and responsibilities, and consequences to know trade-offs when they are applied. The consequences can help to understand and evaluate design pattern before using it. Design patterns can be of simple and composite. When multiple design patterns are combined, they become a single design pattern. The composite design patterns can support the features of more than a single design pattern effectively. Thus composite design patterns can improve the productivity besides configuration management and product derivation. Without merging multiple design patterns it is also possible to have collaboration among design patterns. For instance, Model View Controller (MVC) design pattern can work in tandem with many other design patterns without being combined to make a composite. Database Access Object (DAO) can fill the model part of MVC. In the same fashion, Data Transfer Object (DTO) design pattern can be used to have data transferred from controller to model. A Value Object (VO) design pattern can be used to transfer data from the model to controller. Thus it is evident that design patterns can work together without the need for merging them to form composite ones. In this thesis variability, aware design patterns are used to handle variability in SPL with more ease beside promoting reusability, effective configuration management and quality product derivation.

In computer science ontology is a formal approach for naming and definition of concepts and relationship among them. It is to have knowledge representation effectively. Ontology is identified as one of the approaches that can be used to represent the feature models of SPL in the form of concepts and relationships among them for effective variability management. Feature models of SPL are represented in the form of ontology. As the ontology model is programmable, it is exploited to have dynamic and automatic reconfiguration of SPL [18]. The knowledge representation capabilities and visualization mechanisms of ontology made it be intuitive and user-friendly. Therefore the feature models are formally represented using domain
ontology in order to support change management, improving latency, and automatic reconfiguration without human intervention.

The ontology based feature models are used in this thesis in order to have automatic reconfiguration of SPL. The usage of domain ontology provides a programmable interface in order to help inflexible navigation to various artifacts and products thus realizing dynamic reconfiguration without human intervention. The artifacts are to be reconfigured when changes are made. The change management is possible with domain ontology as it can represent a visible model containing the whole SPL and all features [19]. The feature model transformed into domain ontology has tangible benefits in terms of automating certain activities. Ontology is in fact used for the best representation of knowledge in the real world. The knowledge is represented essentially in the form of concepts. There are relationships between the concepts that are also visualized in the ontology. Ontology usage can be done in any field of computer science. Applications in different fields can exploit ontology to have visibility of knowledge and efficient navigation and manipulation. In this thesis, the author used the ontology for improving efficiency in SPL configuration management and quality product derivation.

1.6 MOTIVATION, OBJECTIVES AND SCOPE OF THE RESEARCH

The software industry has adapted the concept of a family of products known as Software Product Line (SPL) for reusing artifacts. Reusability strategies are one of the cornerstones of SPL. Car manufacturing, mobile device manufacturing, manufacturing of aeroplanes and virtually every area of product development is changed from traditional development to the novel approach of product lines. The software industry is no exception to this. Especially products with longevity are being developed as SPL. In SPL the components involved can be called as artifacts. These artifacts are of two types namely core assets and custom assets. Core assets have fundamental functionality that promotes commonality while custom assets promote variability in SPL. Since variability leads to new products in SPL, variability aware design patterns can help in improving the quality of SPL besides simplifying its configuration management and product derivation. Usage of ontologies also can help
in optimizing configuration management besides promoting visualization of configuration dynamics and even automating the changes in the configuration of SPL.

Efficient configuration management of Software Product Line (SPL) is crucial for the success of any product line software system. Due to ever-changing needs of customers, SPL undergoes constant changes that are to be tracked in real time. As assets in the product line are inter-dependent, changes to them often have a ripple effect and propagate to other products. In the context of customer-driven development, anticipation and change management are to be given paramount importance. It demands implementation of software variability that drives home changed, extended and customized configurations besides economy at scale. Moreover, the emergence of distributed technologies, the unprecedented growth of component based, service-oriented systems throw ever increasing challenges to software product line configuration management. Derivation of a new product is a dynamic process in software product line that should consider functionality and quality attributes. Though configuration management is enough matured for traditional products, very few approaches are found for configuring SPL. They are tailor made and inadequate to provide a general solution. Stated differently, a comprehensive approach for SPL configuration management and product derivation is still to be desired.

**Objectives**

The review of the literature has provided insights that helped the researcher to conceive the following hypotheses that have been tested in this work and results have been analyzed.

- **Hypothesis 1**: A framework for SPL configuration management can improve the quality of configuration management of SPL and derivation of the quality product.
- **Hypothesis 2**: Variability-aware design patterns can exploit feature model of SPL thus leveraging configuration management and product derivation.
- **Hypothesis 3**: Use of ontologies in SPL can effectively represent feature model besides promoting dynamic reconfiguration and quality product derivation.
These hypotheses are the basis for the proposed generic framework with underlying algorithms that exploit variability aware design patterns and ontologies for effective configuration management of SPL and quality product derivation. The first hypothesis is related to have a reusable framework that is the basis for effective configuration management. The second hypothesis is pertaining to the usage of design patterns in SPL configuration management and product derivation. The third hypothesis is related to the usage of ontology to model features and leverage interface for effective configuration management and product derivation.

**Scope of the research**

A generic framework is proposed that is comprehensive and flexible for SPL configuration management and product derivation. The framework has placeholders for variability-aware design patterns and knowledge representation technique such as ontology. It can be adapted to all domains. An algorithm is proposed to provide steps to have configuration management for SPL that implements the change propagation guidelines based on the type of change considered, then the derivation of the new product based on weights, critical path analysis, and feedback. Then design and implementation of variability-aware design patterns and mapping their roles to artifacts is made for optimizing SPL configuration management and also product derivation. The framework is further extended by implementing domain ontology with API for representing feature model in a more formal way. The rationale behind this is that ontology not only represents features of software but also provides flexible means of accessing, altering features and its Metadata. Thus ontology is used to leverage dynamic reconfiguration of SPL and more qualitative product derivation. A prototype application is built to demonstrate the proof of concept. The framework is tested with many SPL case studies including Dr. School product line to demonstrate realization of concepts of the proposed framework in terms of high quality product line configuration and product derivation.

**1.7 ORGANIZATION OF THE THESIS**

The remainder of the thesis is organized into different chapters. Here is the description to help the reader to know briefly about the content of chapters.
Chapter 2

This chapter throws light into the review of the literature on SPL, configuration management of SPL, variability aware design patterns for SPL configuration management and product derivation, and the use of anthologies in improving SPL configuration management and product derivation.

Chapter 3

In this chapter, a generic framework is proposed. It guides SPL configuration and new product derivation in a generic way. It supports three types of changes which are major, minor and micro. Major changes are because of new requirements, minor changes are platform changes and micro changes are bug fixes. Automated Change Propagation algorithm is proposed which supports all the above changes along with 23 cases of change propagations. The framework presented in this chapter is modular and extendable. It has provision for quality product derivation, improving its flexibility by using ontology and variability-aware design patterns.

Chapter 4

This chapter employs methods like weight computation, feedback and critical path analysis for quality product derivation. Two algorithms are proposed in order to achieve this. They are named as Quality Driven Product Derivation (QDPD) and Composition Analysis (CA) algorithms. These are used to make different compositions with core and custom assets and finally find the best composition. Quality product derivation is achieved by using critical path analysis and weights. A prototype is built to demonstrate the proof of concept. The application was tested with Dr. School product line. Results revealed that the proposed framework and underlying algorithms were able to support flexible configuration management and high quality product derivation.

Chapter 5

This chapter focuses on the extension and implementation of the proposed framework in terms of variability-aware design patterns. In the process, identifying variability model and the concept of roles and mapping them to the model is followed. The design pattern roles are mapped to corresponding artifacts in order to realize
variability in SPL with state-of-the-art practices. This has led to the quality and improvement of reconfiguration of SPL. This is evaluated with an application. The empirical study revealed that the implementation of variability-aware design patterns helped improve the performance of configuration management and evolution of new products.

Chapter 6

The focus of this chapter is to propose and implement ontology based feature models for improving configuration management and product derivation. The framework is extended towards this end. It throws light on the use of ontology to achieve dynamic reconfigurations. For reconfigurations, feature model is the model which shows graphically, the similarities and variability of SPL. It has its notations. Feature models are being used to represent variability and similarities in SPL and it helps in new product derivation.

Chapter 7

This chapter provides a summary of the thesis work done besides drawing conclusions and providing recommendations for future work. The summary includes the literature survey insights and the practical work is done right from proposing the framework for efficient configuration management of SPL and quality product derivation. Conclusions provided in this chapter provide insights into the research results.