7-2008

A Model of Open Source Software-Based Product Line Development

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

Faheem Ahmed
Thompson River University, fahmed@tru.ca

Mohammad Ali Babar
Lero, malibaba@lero.ie

A Model of Open Source Software-Based Product Line Development

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

Software Product Line (SPL) and Open Source Software (OSS) have emerged as successful modes of developing software. There is an increased interest in developing suitable approaches for combining the promised advantages of SPL and OSS. Researchers and practitioners have been emphasizing the need of exploiting the ever growing repositories of OSS components for developing SPLs. However, there is no conceptual model for guiding the process of developing OSS-based SPLs. In this paper, we propose a model for developing software product line using open source software. This paper identifies and elaborates the essential phases and activities of the proposed model of developing OSS-based SPLs. This model emphasizes the vital role of software architecture and asserts that software architectures of OSS can be exploited to establish a SPL. To demonstrate this, we have evaluated Eclipse's architecture for its potential to support different flavors of a system.

1. Introduction

A Software Product Line (SPL) is a set of software-intensive systems, which share 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 [1]. A SPL can also be seen as a collection of systems sharing a managed set of features constructed from a common set of core assets and having a significant impact on the software development productivity. A SPL deals with the assembly of products from existing core assets commonly known as components [2], and there is continuous growth in the core assets as the production proceeds [3, 4]. The SPL approach is expected to help organization to reduce cost, improve delivery time and quality by maximize intra-organizational reuse of software artifacts [5, 6]. Another software development paradigm that has recently gained significant attention is Open Source Software (OSS), originated from a pragmatic need to share code among individuals has grown to become a major force behind inter-organizational reuse of platforms, components and code. Several OSS (such as Apache, Linux and Eclipse) have been widely adopted to support mission- and business-critical activities in various sizes of organizations worldwide.

Given the phenomenal success and popularity of both SPL and OSS software development paradigms, researchers and practitioners have been exploring the opportunities and challenges of utilizing the ever growing repositories of shared components provided by OSS in software product lines. It is argued that the use of OSS components in SPL appears to have great potential for both the OSS and SPL communities. For the SPL community, the use of OSS components in a SPL promises to help them to minimize the development efforts in commodity (non-value adding) components. Several OSS components have been successfully used in mission-critical product families [7]. Despite continuously growing interest in finding suitable mechanisms for combing the advantages of OSS and SPL, there is no process guidance model for developing a SPL based on OSS.

We assert that such a process guidance model can help organizations to identify and understand the activities and tasks that need to be undertaken in order to successfully develop OSS based family of systems. In order to address this gap, we propose a model of developing SPL based on OSS by incorporating several concepts that characterizing various aspects of SPL and OSS. The proposed model identifies the interdependency of various activities of SPL and OSS and describes different ways of exploiting the relationships between those activities in order to guide

the process of developing OSS based SPL. It should be clarified that such a process guidance model will not aim to replace existing SPL development and maintenance models and frameworks such as reported in [1, 8]. Rather, this model complements those frameworks for establishing and maintaining SPLs. Since Software architecture and its related issues are considered of paramount importance in the successful development and maintenance of a SPL [9, 10], this model emphasises the vital role of software architecture in developing OSS-based SPL.

2. A Model of Developing OSS-Based SPL

This section presents a model for developing OSS-based SPL. It should be noted that the research underpinning the proposed model does not address the legal and business aspects of using OSS for developing a SPL. To identify the elements of the proposed model, we have drawn upon a number of sources including existing frameworks for establishing and maintaining SPLs as described in [1, 8], an extensive survey of the published literature on software product line engineering, software architecture, and OSS, and an analysis of the heuristics of experienced software architects and SPL researchers and practitioners. However, it is not our intention to claim that this model is complete and fully validated; nor do we assert that it provides an exhaustive list of activities and tasks that an organization is expected to undertake in order to develop SPLs based on OSS. Rather, we expect this model to evolve based on community feedback and empirical assessment that we plan to carry out in our future work. In the following sections, we discuss different elements of the model shown in Figure 1. Before describing each element of the proposed model, it appears quite appropriate to briefly discuss the key role software architecture in SPL. Korhonen and Mikkonen [11] explained that Product Line Architecture (PLA) handles the variations of the applications of some problem domain in multiple abstraction levels, and also guides the developers in the product specialization work. According to Jazayeri et al. [12], PLA defines the concepts, structure, and texture necessary to achieve variation in features of variant products while achieving maximum sharing parts in the implementation. The architectural analysis and design of product lines has been extensively investigated as reported in [10, 13-15].

Meekel et al. [16] identified three axes of variability among products resulting from software product line: features variability, hardware platform variability and performances variability. Features variability describes product specific characteristics. PLA usually contains three major parts, i.e. underlying core architecture, which is the integral composition of all the resulting products from a SPL. Products common features are ones, which are partly or completely present in all the resultant products. Product variable features are ones that are present in individual products. Well-defined core architecture of a SPL is expected to define a trade-off among common and variable features of products that belong to that SPL. We again highlight the important role of architecture in supporting commonalities and variations among different products of a SPL during our discussion on Eclipse architecture's support for SPLs in Section 3.

The Domain Engineering phase of the model establishes an infrastructure for software product line and identifies OSS to be used in developing products, which belong to that SPL. During the Domain Engineering phase, SPL Infrastructure View and OSS Archive View are initiated. The iterations of the activities of SPL Infrastructure View and OSS Archive View provide feedback to one another. The aim is to identify, evaluate, and select suitable OSS components that fulfils the requirements of the SPL and meets the production constraints.

2.1 Product Line Infrastructure View

Product Line Infrastructure View involves conceptualization and initiation of SPL in an organization. This view consists of activities that establish an infrastructure for a SPL. The Product Line Infrastructure View constantly provides feedback to OSS Archive View for effective search, identification, and evaluation, of a potential candidate OSS that can be used to establish a software product line. Software product line scope identifies the characteristics of the product line and the products that comprise the product line. Software product line scope definition activity iteratively provides feedback to OSS search and identification activity in OSS Archive View. This way it ensures that the searched OSS is consistent with the scope of product line. Product line requirements deal with features or functionalities common to all the products belonging to that family. The requirement engineering for product line gives feedback to OSS selection and evaluation activity in the OSS Archive View to find out whether the OSS meets the product line requirements or not. The goals of the software product line are explained by the business cases identified, and they promote the product line. The identification of business cases helps in evaluating identified OSS in the OSS Archive View in order to meet the production criteria and product requirements.

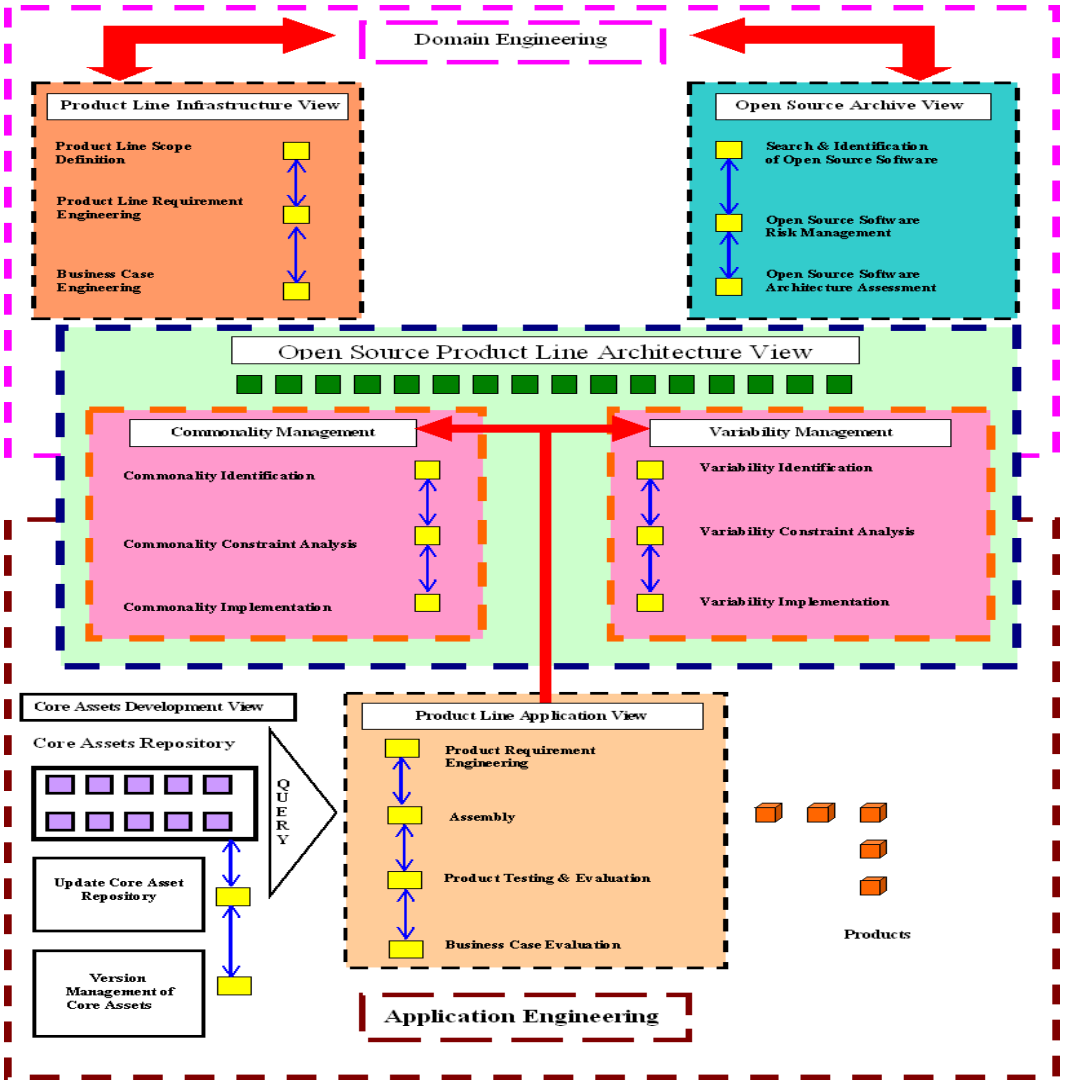

Figure 1: A model of developing Open Source Software based product software product lines

2.2 OSS Archive View

OSS Archive View is responsible for identifying and evaluating OSS for developing SPLs. It communicates with the Product Line Infrastructure View to select a suitable OSS. The evaluation of the OSS is based on the risk management and architectural concerns with reference to a SPL. The process of searching and identifying potential OSS starts when we conceptualize a SPL by defining the product line scope. The main consideration for searching an OSS is to analyse the ability of the OSS for fulfilling the product line requirements and meeting the production constraints, which are considered the most important elements of an evaluation criteria for selecting suitable OSS products based on the guidelines provided in [17]. OSS also introduces some other issues that have to be given appropriate attention before selecting a particular OSS to be used in a SPL. The selection criteria should also take organization's strategies and objectives of using OSS into account. Another important criterion is the architectural level alignment between a SPL and an OSS. That is why evaluating software architecture of an OSS product with regards to the architectural requirements of a SPL is a vital activity. For this purposes, software architecture community has developed several techniques, methods, and tools [10, 15], which can be used for this activity.

2.3 Application Engineering Phase

In the Application Engineering phase of the proposed model (shown in Figure 1), actual products

are developed using OSS components. In this phase, activities of the Product Line Application View interact with the activities of the Core Assets Development View and OSS Product Line Architecture View to produce required products. Product Line Application View initiates requirements of new product and communicates with Core Assets Development View to retrieve required core assets for product development. OSS Product Line Architecture View interacts with Product Line Application View to provide information related to commonality and variability of features based on the product requirement it initiated.

2.4 Product Line Application View

Product Line Application View deals with the actual development of products from open source software. Product Line Application View interacts with Product Line Infrastructure View to identify potential business cases to capture market segment. In order to develop new products Product Line Application View mediates with Open Source Software Product Line Architecture View which maintains the information about core commonality requirements among products and has elaborated extension points in the open source architecture for variability. The assembly activity involves the development of new product. The product requirements guide the assembly process to get feedback from the query activity of Core Assets Development View to find out those potential components suitable to be assembled in order to produce the product. If it is required then assembly activity performs specialization, generalization, or adjustment of the components. Assembly activity introduces variability at the extension points offered by software product line architecture to accommodate the variable part of requirements for a particular product. The qualification criteria of a SPL must be clearly defined so that all the products resulting from that SPL must meet those criteria. In product testing and evaluation, products developed from a SPL are tested to analyse whether they meet the product line testing and evaluation criteria or not. Specific testing and evaluation about integration of components ensures that adaptability has no consequences. Business case evaluation identifies the success and failure story of the products developed and deployed. It compares the proposed business case strategy with the outcome of the development and deployment process of products.

2.5 Core Assets Development View

Core Assets Development View is responsible for providing required components from core assets repository for developing products. Core Assets Development View interacts with Product Line Application View to receive product. In the query activity of the Core Assets Development View, components are searched from the core assets repository in order to develop the product. A well-catalogue core assets repository reduces the efforts to trace the suitable components for assembly. The product requirements serve as an input to the query activity, and continuously traversing core assets repository yields the required components, exactly matched, partially matched or not matched. The components, after adaptation, generate versions, which are documented in this activity. A comprehensive version management and dependency link strategy for components and products in the SPLE provides us with vital information about components and products having a relationship of composition and utilization. A SPL develops an initial core assets repository in the Domain Engineering phase. As a SPL gets matured in its lifecycle, new core assets or even new versions of existing core assets are produced, which must be added to the core assets repository so that they can be reused in later products. The core assets repository is dynamic and continues increasing its size with the addition of new core assets.

2.6 Open Source Product Line Architecture View

The proposed model emphasizes the importance of developing a product line architecture based on OSS product. The junction of Domain Engineering phase and Application Engineering phase produces a suitable product line architecture based on existing OSS components. The Domain Engineering phase provides product line requirements. The Application Engineering phase accommodates those requirements along with product specific requirements. The Application Engineering phase analyses whether the architectures of OSS components meets the characteristics required by the PLA in which those components are supposed to be used. It has been mentioned that a PLA represents the commonalities among the products and variation points where products differ from each other. All the resulting products from a product line share common core architecture.

The software engineering community have proposed several product line architecture design and evaluation methods such as Quality-driven Architecture Design and Analysis method (QADA)

[18] and Family Oriented Abstraction, Specification, and Translation Process (FAST) [3]. One of the commons steps in these methods is the identification of commonality and variability during domain engineering. Variability among products of a SPL is a vital characteristic of software product line engineering. The products of a SPL may vary from each other not only in terms of number and nature of features but also in terms of number and level of required quality attributes such as reliability, security, usability and performance. These variations must be handled systematically to accommodate changes in various products and their different versions belonging to a SPL. 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 a product's requirements. The variability in products can be influenced from internal and external factors. The internal factors have their roots in refining the architecture, whereas the external factors accommodate the market needs and customers' expectations. The introduction of variable features in a product from a software product line is a strategic decision based on market segment [8]. Fitting a component into a 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.

3. Evaluating Eclipse's Architecture

In this section, we present initial findings from evaluating Eclipse's architecture as the proposed model emphasising the importance of exploiting the architectures of OSS for developing SPLs. The main objective of evaluating architecture of Eclipse is to assess its ability to support a SPL development. This activity mainly concentrates on the underlying architecture's ability of supporting the commonality and variability mechanisms required by a SPL. The Eclipse architecture has two main components: runtime platform and Eclipse platform. The runtime platform serves as the underlying core platform for all resulting products. The Eclipse platform is structured around the concept of extension points. Extension points are well-defined places in the system where other tools (called plug-ins) can contribute functionality.

All functionality of the Eclipse platform is a result of interactions between plug-ins and the kernel. Eclipse's architecture is expected to support dynamic inclusion of variability points thus provides a well defined and clear extension points to accommodate variability among products. Plug-ins can define their own extension points or simply add extensions to the extension points of other plug-ins, which illustrates a hierarchical structure of variability points. The platform handles the logistics of the base environment and provides a standard user navigation model. Each plug-in can then focus on doing a small number of tasks to implement a specific set of requirements of a product. Each major subsystem in the Eclipse platform is itself structured as a set of plug-ins that implement some key function and define extension points. Eclipse is written in Java, which makes it a cross-platform application, independent of hardware. Hardware platform variability can be observed in Eclipse due to its platform independent characteristics. Following are the major characteristics of Eclipse architecture, which enables it a potential candidate for software product line architecture:

- Explicit Extension Point: Feature Variability in software products can be introduced by defining plug-ins, which serves as a clear and explicit extension points in Eclipse architecture.
- Hierarchical Structure Plug-ins can extend their functionalities to other plug-ins, thus creating a hierarchy of plug-ins, which makes Eclipse a multi level architecture and allows substantial extensibility keeping commonality among resulting products. Multi level extension allows designers to observe commonality and variability among resulting products.
- Architectural Description Support: Eclipse manifest files provide complete information about the extension points introduced and thus allow designers to understand and analyse the architecture.
- Hardware Variability: Eclipse is a cross platform application thus allows hardware variability to be observed among resulting products.
- Extensible User Interface: Standard Widget Toolkit (SWT) provides an opportunity to develop potable application, which can directly access the user-interface facilities of the underlying operating.

It has also been revealed that although, the Eclipse's architecture has the potential to be used as product line architecture, the quality issues (such as reliability, usability, maintainability and efficiency) need to be given appropriate attention. For example, execution time is one of the major concerns in terms of efficiency of software. If we are developing a SPL,

which has certain execution time requirements, there needs to be suitable mechanisms in Eclipse's architecture to conform to such requirements. Similarly resource allocation and utilization can also be critical issues in software efficiency. For such requirement, one needs to find out whether or not the architecture of OSS (Eclipse in our case) is using the resource allocation and utilization scheme, which is inline with the requirements. Hence, the evaluation of the Eclipse's architecture also revealed that analysing the architecture of OSS from theoretical perspective of SPL in terms of supporting commonality and variability is not sufficient to make a selection decision. Rather, deeper anlaysis should be performed to assess the capabilities of architecture for supporting the required quality attributes in a SPL.

4. Final Remarks

This paper has proposed a conceptual model for open source software-based software product line development. The presented model highlights various activities and tasks that an organization can expect to undertake in order to develop open source software-based SPL. The model has been developed by drawing upon the theoretical principles and industrial practices commonly reported by SPL and OSS communities and discussions with software architecture and SPL practitioners. We assert that this model provides a high level guidance on systematically establishing open source software-based software product line capable of producing multiple products within an application domain. The interdependency of various activities of software product line and open source software captured in the model shows a strong relationship within a common framework of product development. Additionally, the model provides an efficient way of integrating the approaches of software product line and open source software-based development process.

Our future work focuses on identifying suitable techniques and tools from the SPLE, software architecture, and OSS literature for supporting different activities required by the presented model. We also plan to carry out detailed empirical assessment of the utilization and benefits of the model using case study methodology.

5. References

[1] P. Clements and L. Northrop, Software Product Lines: Practices and Patterns. 2002: Addison-Wesley.

[2] M.L. Griss, Implementing Product Line Features with Component Reuse, in Proceedings of the 6th International Conference on Software Reuse. 2000.

[3] D.M. Weiss and C.T. Lai, Software Product Line Engineering: A Family-Based Software Development Approach. 1999: Addison-Wesley.

[4] M.L. Griss, Product Line Architectures, in Component-Based Software Engineering, G.T. Heineman and W.L. Councill, Editors. 2001, Addison-Wesley. pp. 405-419.

[5] G. Buckle, et al., Calculating ROI for Software Product Lines, *IEEE Software*, 2004. **21**(3): pp. 23-31.

[6] F.v.d. Linden, Software Product Families in Europe: The Esaps & Cafe Projects, *IEEE software*, 2002. **19**(4): pp. 41-49.

[7] D. Schmidt, Model Driven Engineering of Product-Line Architectures for Distributed Real-time and Embedded Systems, *Tech Report* Vanderbilt University, USA, 2007.

[8] F.v.d. Linden, K. Schmid, and E. Rommes, Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering. 2007: Springer.

[9] P. Clements and L. Northrop, Software Product Lines: Practices and Patterns. 2001: Addison-Wesley.

[10] J. Bosch, Design & Use of Software Architectures: Adopting and evolving a product-line approach. 2000: Addison-Wesley.

[11] M. Korhonen and T. Mikkonen, Assessing Systems Adaptability to a Product Family, *Journal of System and Software*, 2004. **50**: pp. 383-392.

[12] M. Jazayeri, A. Ran, and F.v.d. Linden, Software Architecture for Product Families. 2000: Addison-Wesley.

[13] D. Paulish, Architecture-Centric Software Project Management. 2002: Addison-Wesley: Reading, MA, USA.

[14] R. Lutz and G. Gannod, Analysis of software product line architecture: An experience report, *Journal of Systems and Software*, 2003. **66**(3): pp. 253-267.

[15] L. Bass, P. Clements, and R. Kazman, Software Architecture in Practice. 2 ed. 2003: Addison-Wesley.

[16] J. Meekel, T. Horton, and C. Mellone, Architecting for Domain Variability, *Proccedings of the 2nd International ESPRIT ARES Workshop on Development and Evolution of Software Architectures for Product Families*, 1998.

[17] D. Cruz, T. Wieland, and A. Ziegler, Evaluation criteria for free/open source software products based on project analysis, *Software Process: Improvement and Practice*, 2006. **11**(2): pp. 107-122.

[18] M. Matinlassi, E. Niemela, and L. Dobrica, Quality-driven architecture design and quality analysis method: A revolutionary initiation approach to a product line architecture, *Tech Report 456*, VTT Technical Research Centre of Finland, Espoo, 2002.