LEVERAGING CONFIGURATION MANAGEMENT AND PRODUCT EVOLUTION OF SPL USING VARIABILITY AWARE DESIGN PATTERNS

By

K.L.S. SOUJANYA * A. ANANDA RAO **

* Associate Professor, Department of Computer Science Engineering, CMRCET, JNTUH, Telangana, India.
** Professor and Director, Department of Computer Science Engineering, JNTUA, Ananthapuramu, Andhra Pradesh, India.

ABSTRACT

Software Product Line (SPL) is an emerging approach to satisfy the ever-increasing customization demands by reusing commonalities and variability's. Variability - aware design patterns can leverage SPL configuration management and evolution of new products. Design pattern is a blueprint or model solution to a frequently occurring design problem. Variability aware design patterns can address variability and help in customizing software products. Modularization of artefacts and reusability of them can be realized by using design patterns. Design patterns in SPL is relatively used in new research area. However, composite design patterns that are variability-aware can lead to the realization of high quality SPL. In this context, the configuration management and product derivation are to be conceived and handled. There are no dedicated efforts found in the literature to leverage the usage of design patterns in SPL. The authors proposed a framework and provided provision for variability-aware design patterns. They use the concept of roles and map them to variability model. Then they map design pattern roles to artefacts thus realizing variability with industry best practices. This will help in improving the dynamic reconfiguration of SPL artefacts. Their empirical evaluation shows that the approach improved performance up to 20% with respect to configuration management of SPL and product derivation. The prototype demonstrates the proof of concept.

Keywords: Variability, Variability-aware Design Patterns, Configuration Management, Software Product Lines, Product Derivation.

INTRODUCTION

Design pattern is a blueprint or model solution to a frequently occurring design problem. While it is useful in reusing the object oriented software components without reinventing the wheel, it also helps in making solutions in time and budget [3]. The term "pattern" was inspired by the work of Alexander Christopher in 1977 in the context of architectural patterns [2]. According to Schmidt [1], there is increasing pressure on development teams to produce quality software. This in fact leads to the reuse of code in both fine grain and coarse grain fashions. The original design patterns proposed by Gamma et al. [4] are classified into creational, structural and behavioural patterns. The creational patterns include Abstract Factory, Factory Method, Builder, Prototype and Singleton. These patterns are used to construct objects in a system independent manner. The structural patterns include Adapter, Bridge, Composite, Decorator, Facade, Flyweight, and Proxy. These patterns are meant for forming large object structures that will simplify design. The behavioural patterns include Chain of Responsibility, Command, Interpreter, Iterator, Mediator, Memento, Observer, State, Strategy, Template Method and Visitor.

There are many instances that show how design patterns revolutionized application development when are used to build frameworks. For instance, Aspect Oriented Programming (AOP) paradigm is based on Decorator design pattern, while Java's SWING is based on Observer and Model View Controller (MVC) design patterns. Design patterns play a vital role to achieve the actual implementations that can be reused with ease [2]. Variability-aware design patterns and concepts were explored in [5], [6], [7], [8], [9], [10] and [11].

Having understood the significance of design patterns, in this paper the authors explored the usage of variability-aware design patterns in SPL configuration and derivation of new product. Their contributions include the design and implementation of variability-aware design patterns and mapping their roles to artefacts. This will help in optimizing SPL configuration management and also affect product derivation. The remainder of the paper is structured as follows. Section 1 on Related work. Section 2 throws light into variability-aware design patterns. Section 3 presents the optimization of SPL configuration management though variability aware design patterns. Section 4 explains the evaluation procedure. Section 5 concludes the paper besides providing directions for future work.

1. Related Work

Variability-aware design patterns and their usage were found in the literature, while the effects of that on configuration management of SPL and product evolution are explored in this paper. Becker [5] proposed a model for variability in SPL to manage variability systematically and efficiently. Parra et al. [6] explored AOP and design patterns to enhance SPL and runtime adaptation. Tizzei et al. [7] also studied AOP for assessing design stability of SPL by comparing OO and AO approaches. Fortier et al. [8] proposed design patterns for dealing with variability in SPL pertaining to mobile software. They studied the impact of variability in different domains. Hammouda et al. [9] studied feature-driven variability and design patterns for separation of concerns for better change management.

Alves et al. [10] analyzed variability and commonality in variation management between Runtime Adaptable System (RTA) and SPL. Schuster [11] employed role modelling for pattern-based SPL design. The roles present in a variability-aware design pattern are mapped to elements of artefacts in SPL. This work is closely related to the authors work in this paper. In [11], the variability-aware design patterns were explored for enhancing SPL. However, the work in this paper focuses on leveraging configuration management and product evolution in an SPL through variability-aware design patterns. The focus of this paper is to optimize performance of configuration management and product derivation.

2. Variability-Aware Design Patterns and Role Modelling

Before entering into the subject of variability-aware design patterns and role modelling for improved derivation of new product and effective configuration management of SPL, the authors provide the context in which the paper proposed about variability aware design patterns.

2.1 Overview of the Proposed Framework

The authors proposed and implemented a framework that has provision for configuration management, and product derivation. Their focus in this paper is to investigate how the variability-aware design patterns can contribute to the improved configuration management of SPL and effective product derivation. This is the rationale behind showing the framework (it was proposed by the authors in their previous paper titled "A Generic Framework for Configuration Management of SPL and Controlling Evolution of Complex Software Products" [12]) in Figure 1 for completeness.

Therefore, the focus of this paper is about the variability-

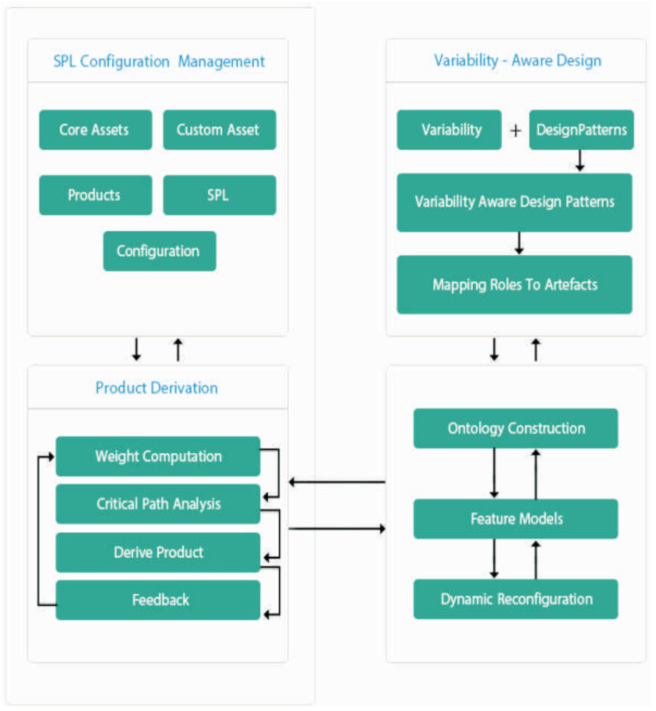


Figure 1. Overview of Proposed Framework

aware design pattern. More information on the framework can be found in their previous paper [12].

2.2 Variability – Aware Design Patterns

In the context of SPL, commonalities and variability's are the two things that are exploited. However, there are certain challenges that are thrown to software engineers when dealing with SPL. The challenges can recur and need best practices or design patterns to overcome the issues. Design patterns are the proven solutions to recurring design problems in object oriented software development. Now before knowing variability-aware design patterns, variability is defined. Variability refers to the differences between the products in the SPL. The differences between products in SPL can also be understood in terms of features. Therefore differences in features can be attributed to variability. As products in a family of products tend to vary, the variability is reflected. Due to the variability's and the dynamic nature of contemporary software requirements, it became indispensable to consider variability's in SPL and handle them. They cannot be ignored and to be taken into account. This paper throws light into the advantages of considering variability-aware design patterns and how they can cater to improved configuration management of SPL and effective product derivation.

In the framework presented in Figure 1, a placeholder is for "variability-aware design patterns" which can help in handling variability's in SPLs effectively. Having understood the variability, let us know more about variability-aware design pattern. It is the design pattern that can handle variability's so as to reduce design effort and time. It also can lead to simplified configuration management product derivation. This hypothesis is tested in this paper. The details in the subsequent sections provide the dynamics of this proposition. In other words, the ability to customize or change a system is known as variability [13]. Since SPL is to maximize the reuse of commonalities and variability's besides simplifying customization, variability-awareness is essential.

2.3 Managing Variability

In SPL, it is very important to keep the future changes in mind while designing systems. It is not easy to support variability unless the architecture is flexible and can adapt to new features. There are many aspects involved in managing variability. They are identifying variability, introducing variability into the system, collecting the variants, and binding the system to one variant. First of all, the developers need to understand the changing requirements and incorporate them in the requirements specification of SPL. In this process, they need to identify the variability's based on the proposed customizations or changes. Once it is done, the variability's are to be implemented in order to pave way for a customized product. There are many variants associated with one variability point. Such variants are to be collected. When developers collect variants, the system can adapt to different variants based on the requirement. Finally, the system has to be bound to one particular variant. Again the binding can be done internally or externally based on the component in which functionality is implemented. Often the configuration management tools perform the binding.

Table 1 shows the three patterns viz the variant entity, optional entity and multiple coexisting entity pertaining to variability and the characteristics of them which are compared that includes feature diagram, management, scope of binding, collection, binding and open and closed. Variant entity and optional entities can be determined by the developers, while the multiple coexisting entity needs system to be processed to choose between the available variations.

Characteristic	Variant Entity	Optimal Entity	Multiple Coexisting Entity
Feature Diagram	XOR branch	Optimal feature	Or branch
Management	Separate from use	Separate from use	Performed for every use
Scope of Binding	Valid for entire system	Valid for entire system	Valid for one use
Collection	Implicit or Explicit	Not Applicable	Explicit
Binding	External or Internal	External or Internal	Internal
Open and Closed	Depends on Runtime Environment	Immediately Closed	Depends on Runtime environment

Table 1. Three Patterns Pertaining to Variability and their Characteristics (excerpt from [13])

2.4 Role Modelling

In the context of object oriented programming, a solution is made in terms of objects and the interactions among the objects. Having said this, it is true that an object in the real world can play different roles. Due to collaborations with other objects, they can exhibit the different roles they can play. Thus the concept of role modelling is emerged. This has been used in this paper to adapt it to variability-aware design patterns that can simplify SPL, its configuration and product derivation. Role modelling can build the gap between concrete design (class diagram for instance) and a design idea (with collaborations in mind) [11]. Roles can capture true dynamics of real objects instead of a static structure [14]. The following notations are used in role modelling.

As shown in Table 2, the notations are presented and they are used in the subsequent sections to describe how the role modelling can be employed to variability-aware design patterns.

2.5 Defining a Variability Aware Design Pattern

When design patterns encapsulate variability's, they are known as variability-aware design patterns. On the other hand, role modelling can help identify collaborating parts in design patterns. A family of role models can be captured and used for describing variability-aware design patterns. By combining two role models, a family of role models can be derived. Here is an example for variability-aware design pattern (Figure 2), which is a hybrid design pattern, namely Adapter Facade, which combines Adapter and Facade patterns is proposed by

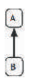
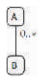
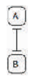
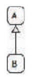
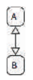
Notation	Description	Example
Use	Indicates that an object playing certain role uses another object playing different role.	
Association	Two objects playing different roles are known to each other.	
Prohibition	Two objects are exclusive in playing two different roles.	
Implication	It implies that object playing role A should also play role B.	
Equivalence	It implies that object playing role A should also play role B and vice versa.	

Table 2. Shows Notations used for Role Modelling

Gamma et al. [4]. The Adapter Facade pattern thus proposed exploits inheritance for adapting incompatible classes besides unifying interface for client. In the next section, the authors relate this pattern with role modelling and mapping the roles to artefacts.

2.5.1 Intent

This design pattern converts incompatible interface of a class into another interface, thus enabling incompatible classes to work together besides providing a unified and high level interface to a sub system of interfaces. Thus it is can adapt between classes and provide a simplified interface to a class library.

2.5.2 Motivation

There are cases where two classes need to have interaction. However, they could not interact due to subtle differences in interfaces. In other words, they are not compatible with each other. Yet they have functionality that can be reused and the problem is with interfacing. Another case is that, there are many classes with different interfaces. A unified interface is lacking thus making it unduly complex. In this case, a design pattern that can play the dual role of removing incompatibility and bestowing a simplified interface to a sub system of classes is very handy from the design perspective.

Consider for example, a customer wants to book tickets for his travelling requirements. He approaches a travel agent who has required interface for booking flight tickets locally and internationally. The interface is defied in a pure abstract class called Booking Agent which has incompatibility with other classes like United States. The booking sub system of USA has plethora of interfaces that are so complex. In this context, how can the classes that have incompatible classes work together? And how can the USA bookings sub system interfaces can be simplified or unified? To achieve this, new design pattern (Adapter Facade) is defined. In travel booking example, the design pattern plays dual role of playing a facade (coarse grained interface) to USA travel sub system and being adaptive to incompatible interfaces of the class United States.

2.5.3 Structure

The consequences of the design patters are described

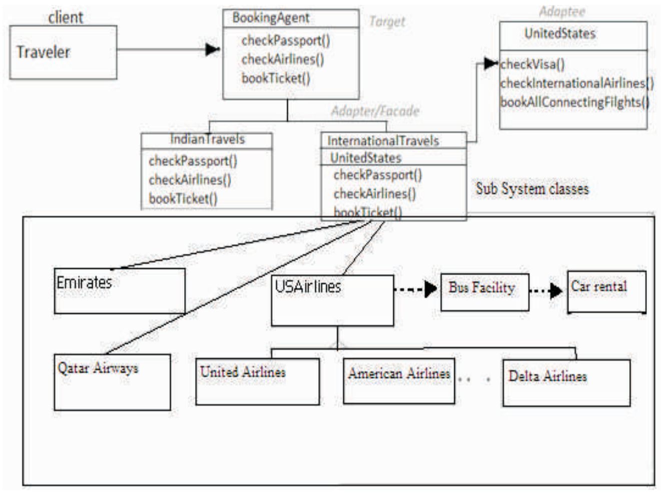


Figure 2. Composite Design Pattern Named Adapter Facade

here. Adapting sub classes of Adaptee is not possible. It might override the Adaptee's behaviour. A single instance adapter is used with having any pointer for indirection. It shields clients from sub system. It does mean that the subsystem is transparent to sub system. Its complexity is made simple. It promotes loose coupling between clients and the sub system. The components in the sub system might be strongly coupled. Complex and circular dependencies are avoided. Reduces compilation dependencies, thus making very complex systems look simple. The design pattern doesn't force to avoid the usage of sub system directly. This pattern can be used, when it is required to reduce the coupling of clients with sub system by making a unified interface with course grained calls to diversified and fine grained sub system interfaces and eliminate the incompatibilities between classes with similar and compatible functionalities.

A pattern has number of roles associated with it. The roles are in turn bound to the system elements that are part of various artefacts. A hybrid design pattern such as Adapter Facade described earlier can have a family of roles that can be associated with many artefacts. A variability-aware design pattern is a collection of software elements that separate a concern in SPL.

2.6 Modelling Adapter Facade Design Pattern Using Role Modelling

With respect to reservation system SPL, the role modelling and mapping the roles to artefacts is described in this sub section. The role diagram presented in Figure 3 provides the roles and the relationships among them. These roles belong to the reservation system. The Adapter Facade design pattern is variability aware and the roles in the system are having certain relationship with other roles. The possible relationships as given in Table 2 are Use, Association, Prohibition, Implication, and Equivalence.

The roles identified are client, target, facade, adapter, adaptee and subsystem. These roles are actually taken from the simplified system for brevity. The client role has Use relationship with target role. In the same fashion, the facade role has Use relationship with subsystem role. The facade and adapter have implication relationship with target role. The adapter and adaptee roles do have prohibition relationship between them. These relationships are as per the role modelling concept that is pertaining to the variability-aware design patterns.

As shown in Figure 4, the client role is mapped to Traveler, which actually initiates the reservation process. The target role is mapped to the Booking Agent, which is actually the interface which is responsible for all reservation functionalities. The Facade role is associated with International Travels which acts as a facade to many sub system classes. This will make many fine grained calls to the subsystem to achieve reservation job. The facade works as the simplified interface between the Booking Agent and subsystem classes. Facade avoids many round trip calls to server in order to improve performance. The adapter role is associated with International Travels class. It is responsible to convert the incompatible interface present inthe United States class into a compatible interface. Thus the adapter class makes it

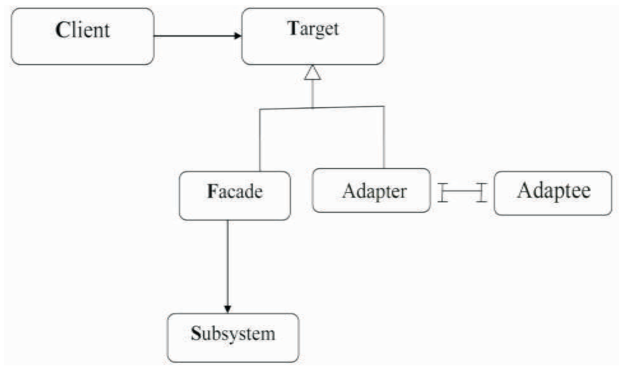


Figure 3. Role Diagram for Reservation System

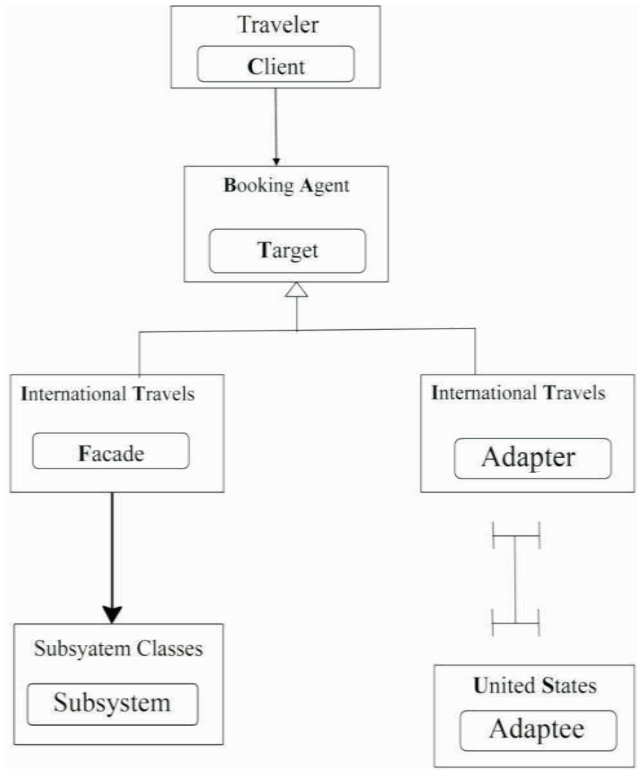


Figure 4. Mapping of Role Modelling to Class Hierarchy in Reservation System SPL

possible to integrate with other systems and handle variability. The adaptee role associated with the United States class is responsible to have reservations with flights operated from the United States of America. The role modelling and mapping results in simplification of SPL configuration management besides improving the performance of product derivation. These two claims are evaluated with human experts as described in section 4.

3. Variability-aware Design Patterns for Optimizing Configuration Management and Product Derivation

Variability-aware design patterns can help in handling cross-cutting concerns. As such, patterns comprise many roles that are mapped to the elements of artefacts in SPL and the configuration management of SPL get optimized. All elements that are bound to a role can be easily configured and managed. The usage of variability-aware design patterns can not only separate cross-cutting concerns, but also help in managing configuration of artefacts of SPL with ease. The productivity in SPL configuration management and the accuracy in the derivation of new product are improved.

The hybrid design pattern presented in Figure 2 plays an adapter role and also facade role besides other roles. In reservation system SPL the configuration management has been simplified due to the usage of variability-aware design patterns. The role modelling and mapping roles to artefacts was described in the previous section. The authors have built a prototype application that demonstrates the proof of this concept. The prototype has provision for mapping roles into elements of artefacts thus making the cross cutting concerns to be managed with ease. The versioning and reconfiguration is made simple. The product derivation is also proved to be consistent and accurate.

The prototype has features that help to configure artefacts that contain elements which are mapped to the roles in design patterns. The mapping of roles to elements of artefacts optimizes the process of configuration management. This is due to the fact that the cross cutting concerns are separated in the form of variability-aware design patterns. This has improved the flexibility and configuration management of the prototype along with product derivation.

3.1 Case Studies Considered

Reservation system, Dr. School and Library Management System (LMS) are the three SPLs considered for experimentation. These case studies are supported by the prototype that were developed using Java programming language. The prototype demonstrates the configuration management and product derivation. In this paper, the authors have provided the details of Reservation System in terms of variability aware design pattern usage, role modelling and roll mapping. However, the implementation and evaluation was done for all three SPLs. Due to space constraint, Dr. School and LMS could not be elaborated. However, the evaluation results are presented in the subsequent section.

4. Evaluation

The evaluation methodology used has been described here. They have evaluated the proposed approach with different SPLs. This prototype application with and without variability-aware design patterns for SPLs were given to 15 human experts. These experts do have experience in software engineering methodologies and working knowledge in SPL design and implementation. They have expertise in design patterns as well. These experts are from different software companies located in India. They approached more than 25 experts who were working for different organizations. And 15 members have accepted their request to evaluate this prototype. Out of 15 candidates selected, only 10 members could participate in the evaluation. Their main focus was to evaluate the variability-aware design patterns usage in SPL. They conducted experiments with three SPLs to know the impact of variability-aware design patterns on SPL configuration management and product derivation. Their study is two-fold in nature. First, they did experiments using prototype without variability-aware design patterns (PROTOTYPE I). Second, they did experiments using prototype with variability-aware design patterns (PROTYPE II). Both sets of experiments were on configuration management and product derivation efficiency.

SPL configuration management is evaluated in terms of the time taken, number of updates and ease of retrieval. Product derivation is evaluated in terms of accuracy in derivation of product with high quality artefacts. Their findings are given in terms of percentage difference between the Prototype I and Prototype II. The percentages given by 10 experts are taken and average percentage is found. The evaluation results of their observations on three SPLs are as shown in Figure 5 and Figure 6.

As seen in Figure 5 and Figure 6, there is performance

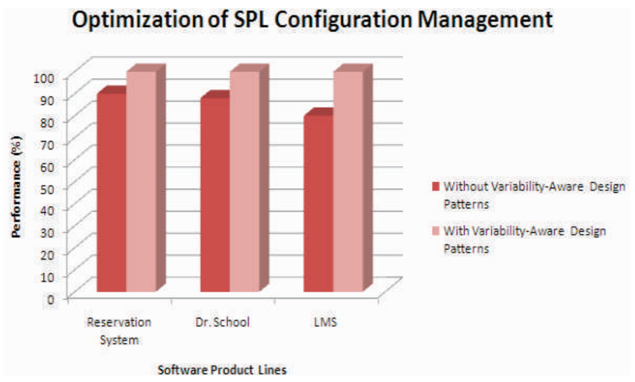


Figure 5. The Average of Performance % given by Human Experts (Configuration Optimization)

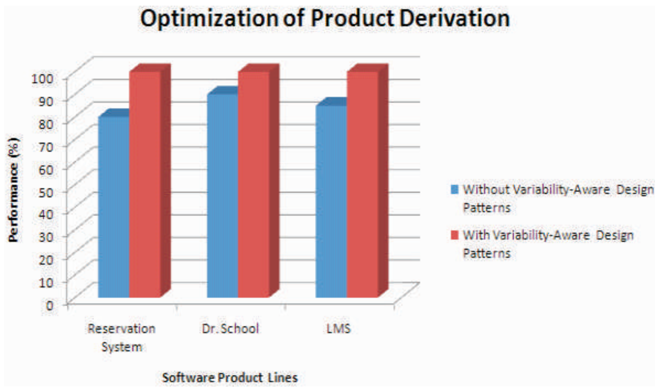


Figure 6. The Average of Performance % given by Human Experts (Product Derivation Optimization)

improvement in both configuration management and product derivation. This is from the view point of human experts who were involved in the evaluation of the prototype that incorporated variability-aware design patterns. There is a significant percentage difference between the PROTOTYPE I and PROTOTYPE II with respect to configuration management and product derivation.

Conclusions and Future work

In this paper, the authors incorporated variability-aware design patterns into configuration management and product derivation, which is the main prototype. The idea behind this was to exploit the concept of roles in design patterns that can be mapped to elements of artefacts for optimization of SPL besides product derivation and configuration management. The authors evaluated their prototype with three SPLs such as reservation system, Dr. School and LMS. The evaluation was carried out by select human experts.

They evaluated PROTOTYPE I and PROTOTYPE II on the three SPLs and provided their performance observations in terms of product derivation and configuration management. The empirical results revealed that the usage of variability-aware design patterns improved the performance of SPL configuration management and product derivation. The performance is improved by up to 20 percent. This research can be extended further to improve this prototype for incorporating ontology's for real time reconfiguration of SPL and its artefacts.

Acknowledgements

The authors thank the reviewers at ROSE lab, JNTUA, Anantapuramu and CMRCET for providing facilities to carry out this research work. They would also like to thank all the experts from different software companies who participated in the evaluation of this SPL configuration management prototype.

References

[1]. **Schmidt, D. C.** *Design Patterns to Develop Object-Oriented Communication Software Frameworks and Applications,* (n.d), pp.1-16.

[2]. **Blaimer, N., Bortfeldt, A. and Pankratz, G. (2010).** *Patterns in Object-Oriented Analysis*. fern universittat in hagen. (n.d), pp.1-80.

[3]. **McDonald, J.,** Design Patterns. *Dzone*. (n.d), pp.1-7.

[4]. **Gamma, E., Helm, R. Johnson, R., and Vlissides, J. (1994).** *Design patterns: Elements of Reusable Object-Oriented Software*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.

[5]. **Becker, M. (2003).** "Towards a General Model of Variability in Product Families". *System Software Group*, University of Kaiserslautern, pp.859-876.

[6]. **Parra, C., Blanc, X., Cleve, A., and Duchiena, L. (2011).** "Unifying design and runtime software adaptation using aspect models". *Elsevier*. Vol.76 , pp.32-44.

[7]. **Tizzei, L. P., Dias, M., Rubira, C. M. F., Garcia, A., and Lee, J. (2011).** "Components meet aspects: Assessing design stability of a software product line". *Elsevier*. Vol.53, pp.25-34.

[8]. **Fortier, A., Rossi, G., Gordillo, S. E., and Challiol, C. (2010).** "Dealing with variability in context-aware mobile software". *Elsevier*. Vol.83, pp.737-772.

[9]. **Hammouda, I., Hautamaki, J., Pussinen, M., and Koskimies, M. (2005).** "Managing Variability Using Heterogeneous Feature Variation Patterns". *Springer-Verlag,* Berlin, Heidelberg, pp.977-1000.

[10]. **Alves, V., Schneider, D., and Becker, M. (2009).** "Comparitive Study of Variability Management in Software Product Lines and Runtime Adaptable Systems". *ACM*, pp.4212-4233.

[11]. **Schuster, S. (2014).** *Pattern-Based Software Product Line Design using Role Modeling*. pp.1-136.

[12]. **K.L.S. Soujanya and A. Ananda Rao, (2015).** A "Generic Framework for Configuration Management of SPL and Controlling Evolution of Complex Software Products". *ACM SIGSOFT Software Enginering Notes*, Vol.41, No.1, January 2016.

[13]. **Jilles Van Gurp, Jan Hosch, and Mikael Svahnberg (n.d).** *Managing Variability in Software Product Lines*. Retrieved from http://www.jillesvangurp.com/static/managingvariabilityinSPLs.pdf

[14]. **D. Riehle, (1997).** "A role-based design pattern catalog of atomic and composite patterns structured by pattern purpose". *Tech. rep. Ubilab Technical Report 97.1. 1*. Zürich, Switzerland: Union Bank of Switzerland.

ABOUT THE AUTHORS

K.L.S. Soujanya is an Associate Professor in the Department of Computer Science Engineering at CMRCET, JNTUH, Telangana, India. She received her B.E Degree from Osmania University, Hyderabad, Telangana, India and M.Tech Degree in CSE from JNTU College of Engineering, Anantapuramu, Andhra Pradesh, India. She is currently pursuing her Ph.D at JNTUA, Anatapuramu, Andhra Pradesh, India. She has also attended various Conferences at IIIT Hyderabad, IIT Chennai, Infosys Mysore and workshops at JNTUA, JNTUH. Her research areas include Software Engineering, Cloud Computing and Data Mining.

Dr. Ananda Rao Akepogu received his B.Tech Degree in Computer Science and Engineering from University of Hyderabad, Andhra Pradesh, India and M.Tech Degree in A.I & Robotics from University of Hyderabad, Andhra Pradesh, India. He received his Ph.D Degree from Indian Institute of Technology Madras, Chennai, India. He is a Professor of the Department of Computer Science and Engineering and currently working as Director of industrial relations and placements at JNTUA College of Engineering, Anantapur, Jawaharlal Nehru Technological University, Andhra Pradesh, India. Dr. Rao has published more than 100 publications in various National and International Journals/Conferences. He received Best Research Paper award for the paper titled "An Approach to Test Case Design for Cost Effective Software Testing" in an International Conference on Software Engineering held at Hong Kong, 18-20 March 2009. He also received Best Educationist Award, Bharat Vidya Shiromani Award, Rashtriya Vidya Gaurav Gold Medal Award, Best Computer Teacher Award and Best Teacher Award from the Andhra Pradesh Chief Minister for the year 2014. His main research interest includes Software Engineering and Data Mining.