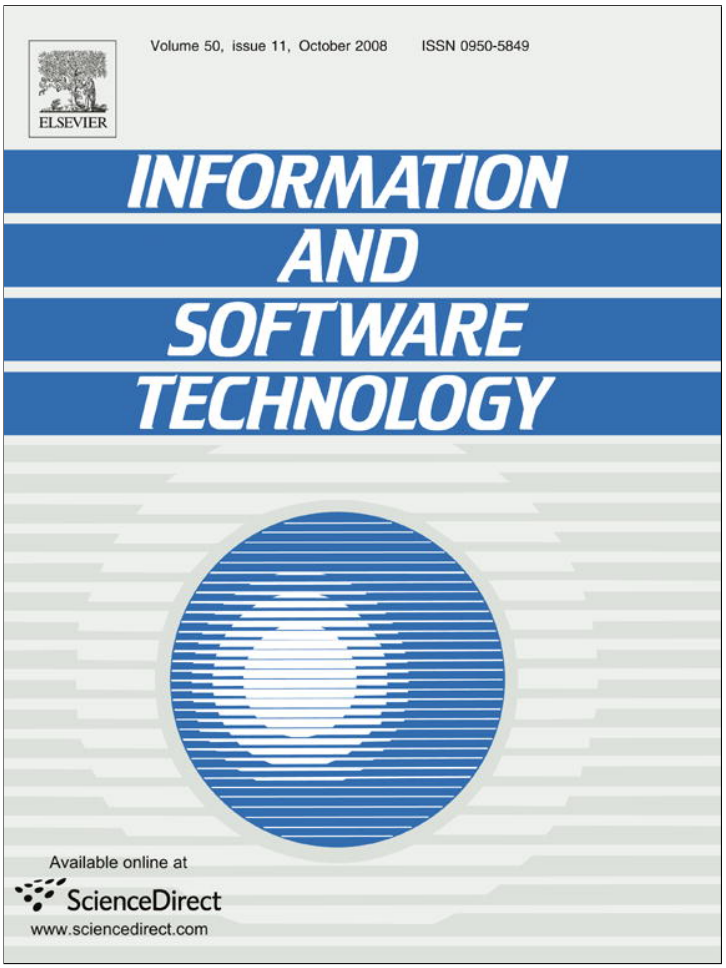

Available online at www.sciencedirect.com
ScienceDirect
Information and Software Technology 50 (2008) 1098–1113
INFORMATION AND SOFTWARE TECHNOLOGY
www.elsevier.com/locate/infsof

The software product line architecture: An empirical investigation of key process activities

Faheem Ahmed [a,*], Luiz Fernando Capretz [b]

[a] College of Information Technology, P.O. Box 17555, United Arab Emirates University, Al Ain, United Arab Emirates
[b] Department of Electrical & Computer Engineering, Faculty of Engineering, University of Western Ontario, London, Ont., Canada N6A 5B9

Received 25 April 2007; received in revised form 18 October 2007; accepted 23 October 2007
Available online 30 October 2007

Abstract

Software architecture has been a key area of concern in software industry due to its profound impact on the productivity and quality of software products. This is even more crucial in case of software product line, because it deals with the development of a line of products sharing common architecture and having controlled variability. The main contributions of this paper is to increase the understanding of the influence of key software product line architecture process activities on the overall performance of software product line by conducting a comprehensive empirical investigation covering a broad range of organizations currently involved in the business of software product lines. This is the first study to empirically investigate and demonstrate the relationships between some of the software product line architecture process activities and the overall software product line performance of an organization at the best of our knowledge. The results of this investigation provide empirical evidence that software product line architecture process activities play a significant role in successfully developing and managing a software product line.

Keywords: Software product line; Software architecture; Empirical study; Software engineering; Domain engineering

1. Introduction

Software architecture has a history of evolution and over a decade the software industry is observing and reporting refinements and advancements. Now the trends in software architecture for single product development have turned into software product line architecture for line of resulting products. Software product line is a set of software-intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment and are developed from a common set of core assets in a prescribed way [11]. The software product line is increasingly gaining the attention of software development organizations because of the promising results on cost reduction, quality improvements and reduced delivery schedule. Clements et al. [10] report that software product line engineering is a growing software engineering sub-discipline, and many organizations, including Philips®, Hewlett-Packard®, Nokia®, Raytheon®, and Cummins®, are using it to achieve extraordinary gains in productivity, time to market, and product quality. The economic potentials of software product line have been recognized in software industry [6,60].

There are other corresponding terminologies for software product line, which have been widely used in Europe, for example ''product families'', ''product population'', and ''system families''. The acronym BAPO [61] (Business-Architecture-Process-Organization) defines process concerns associated with software product lines. Business, architecture, process and organization are considered critical because they establish an infrastructure and manage the way the products resulting from software product lines make profits. The architecture dimension of the software product line concept attained most of the attention of researchers and the architecture aspects of software product lines such as domain engineering, product line architecture, commonality and variability management has been a

* Corresponding author. Tel.: +971 50 9357086; fax: +971 3 762 6309.
E-mail addresses: f.ahmed@uaeu.ac.ae (F. Ahmed), lcapretz@eng.uwo.ca (L.F. Capretz).

0950-5849/$ - see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.infsof.2007.10.013

key area of research since the introduction of the concept after mid nineties. Research has been reported on software product line process methodology including, product line architecture, commonality and variability management, core assets management, business case engineering, application and domain engineering [5,12,32,65].

This paper's main contribution is to increase the understanding of the influence of some of the key architecture process activities by showing empirically that they play an imperative role in the better performance of software product line within an organization. A quantitative survey of software organizations currently applying the concept of software product lines over a wide range of operations, including consumer electronics, telecommunications, avionics, and information technology, was designed to test the conceptual model and hypotheses of this investigation. This study provides empirical evidence that software product line architecture process activities play a significant role in successfully developing and managing a software product line.

1.1. Software product line architecture: related work

Software architecture has been a key research area in software engineering due to its profound impact on the productivity and quality of software. Software architecture is the structure of the components of a program or system, their interrelationships, and the principles and guidelines governing their design and evolution [29]. Software architecture has a long history of evolution and in this modern age this transformation leads towards software product line architecture, where the concern is not a single product development rather the focus is on multiple product development by sharing the same architecture. Pronk [51] defines software product line architecture as an ultimate reuse in which the same software in reused for an entire class of products with only minimal variations to support the diversity of individual product family members. According to Jazayeri et al. [32] software product line architecture defines the concepts, structure, and texture necessary to achieve variation in features of variant products while achieving maximum sharing parts in the implementation. Mika and Tommi [46] further elaborate that software product line architecture can be produced in three different ways: from the scratch, from existing product group, or it can be build from a single existing product. Software product-line architecture is a powerful way to control the risks and take advantage of the opportunities of complex customer requirements, business constraints, and technology, but its success depends on more than technical excellence [21]. The software product line architecture captures the central design of all products and allows for the expression of variability and commonalities of the product instances, the products are instantiated by configuring the architecture and customizing components in an asset library [63]. The ''Architecture'' in BAPO is considered critical because it deals with the technical means to build an architecture that is aimed to share by a number of products from the same family. van der Linden et al. [61] identify some main factors in evaluating the architecture dimension of software product line such as: software product family architecture, product quality, reuse levels and software variability management and classified the architecture maturity of software product line into five levels in the ascending order: independent product development, standardized infrastructure, software platform, variant products and self-configurable products. Birk et al. [4] conclude that explicit documentation of the software product line architecture, platform features, and generic interfaces is important for the product teams to understand the reusable assets.

The methodologies developed for software product line development either in general or specific to particular application domain consider domain engineering as an integral activity of the overall product line process and has profound impact on building the architecture for the product line. Bayer et al. [3] at Fraunhofer Institute of Experimental Software Engineering (IESE) develop a methodology called PuLSE (Product Line Software Engineering) for the purpose of enabling the conception and deployment of software product lines within a large variety of enterprise contexts. PuLSE-DSSA is a part of PuLSE methodology, which deals with developing the reference architecture for software product line. Knauber et al. [40] further elaborates that the basic idea of PuLSE-DSSA is to incrementally develop reference architecture guided by generic scenarios that are applied in decreasing order of architectural significance. Researchers at Philips® developed Component-Oriented Platform Architecting (CoPAM) [1] method for the software product lines of electronics products. CoPAM assumes a strong correlation among facts, stakeholder expectations, any existing architecture and the institutions about possible architects in developing software product line architecture. Weiss and Lai [65] discuss the development of Family-Oriented Abstraction Specification and Translation (FAST) method for software product line process and successful use at Lucent Technologies®. FAST method covers a full software product line engineering process with specific activities and targeted artifacts. It divides the overall process of software product line into three major steps of domain qualification, domain engineering and application engineering. Researchers at IESE developed a methodology called KobrA [2], which defines software product line engineering process with activities and artifacts. The process of software product line engineering is divided in to framework engineering and application engineering with their sub steps. These steps cover the implementation, releasing, inspection and testing aspects of product line engineering process. Kang et al. [36] propose a Feature Oriented Reuse Method (FORM), which is an extension to the Feature-Oriented Domain Analysis (FODA) method to cover the aspects of software product lines. FORM provides a methodology to use feature models in developing domain architectures and components reusability. VTT technical research centre of Finland develop Quality-driven Architecture Design and Quality Analysis (QADA)

method for developing and evaluation of software architecture with emphasis on product line architecture. Matinlassi [45] reported the comparison of software product line architecture design methods including CoPAM, FAST, FORM, KobrA and QADA, and concluded that these methods do not seem to compete with each other, because each of them has a special goal or ideology.

The concepts of commonality and variability management inherently belongs to domain engineering are gaining popularity over time due to extensive involvement in software product line concept. According to Coplien et al. [16] commonality, and variability analysis gives software engineers a systematic way of thinking about and identifying the product family they are creating. Commonality management deals with the way features and characteristic that are common across the products belong to same product line whereas variability management is other way round. Variability management handles the way the variable features and characteristic are managed in different product of the same product line. Software product line requires systematic approaches to handle commonality and variability and the core of successful software product line management largely rely on effective commonality and variability management. Kang et al. [36] discuss the use of feature models to manage commonality and variability in software product line. Lam [41] presents variability templates and variability hierarchy based variability management process. Thompson and Heimdahl [57] propose a set based approach to structure commonalities and variability in software product lines. Kim and Park [38] describe the goal and scenario driven approach for managing commonality and variability on software product line. Ommering [59] observes that the commonalities are embodied in an overall architecture of software product line, while the differences result in specifying variation points and by filling those variation points, individual products can be derived. Other researchers [40,44,65] stressed that the software architecture for a product family must address the variability and commonality of the entire set of products.

Requirements modeling have always been a key architecture concern in software devolvement, because it provides a better understanding of the requirements of the architecture and allows visualizing the interconnection of various sub-units. Since the popularity of object oriented design, Unified Modeling Language (UML) has become an industry standard, many researchers attempt to introduce UML in visual modeling of software product line architecture by presenting enhancement in the current state. Birk et al. [4] stress that the organization dealing with software product line architecture should describe the architecture using well-established notations such as the UML and the architecture description should cover all relevant architectural views and use clearly defined semantics. Gomma and Shin [30] describe a multiple-view meta-modeling approach for software product lines using the UML notation, which defines the different aspects of a software product line such as: the use case model, static model, collaboration model, state chart model, and feature model. Zuo et al. [67] present the use of problem frames for product line engineering modeling and requirements analysis and demonstrate some additional notation to support the requirements management and variability issues in product line problem frames. Dobrica and Niemelä [22] discuss how UML standard concepts can be extended to address the challenges of variability management in software product line architecture and introduce some extensions in UML standard specification for the explicit representation of variations and their locations in software product line architectures, this work is based on Quality-driven Architecture Design and quality Analysis (QADA) methodology. Eriksson et al. [24] describe a product line use case modeling approach named PLUSS (Product Line Use case modeling for Systems and Software engineering) and conclude that PLUSS performs better than modeling according to the styles and guidelines specified by the Rational Unified Process (RUP) in the current industrial context.

Software architecture evaluation techniques are generally divided into two groups of qualitative evaluation and quantitative evaluation. Qualitative techniques include scenarios, questionnaires, checklists etc. Quantitative techniques cover simulations, prototypes, experiments, mathematical models, etc. Etxeberria and Sagardui [25] highlight the issues that can arise when evaluating product line architecture versus evaluating single system architecture, including classifications of relevant attributes in product line architecture evaluation, new evaluation moments and techniques. Graaf et al. [31] present a scenario based software product line evaluation technique, which provides guidelines to adapt scenario-based assessment to software product line context. Using the qualitative technique of software architecture evaluation van der Hoek et al. [58] put forward service utilization metrics to assess the quality attribute of software product line architecture. Zhang et al [66] study the impact of variants on quality attributes using a Bayesian Belief Network (BBN) and design a methodology applicable to software product line architecture evaluation. De Lange and Kang [20] propose a product-line architecture prototyping approach using network technique to assess issues related to software product line architecture evaluation. Gannod and Lutz [28] define an approach to evaluate the quality and functional requirements of software product line architecture. Niemelä et al. [47] discuss the basic issues of product family architecture development and presents evaluation model of software product family in industrial setting.

The literature survey of the related work of software product line architecture exposes some key architecture process activities such as domain engineering, commonality and variability management, requirements modeling, architecture documentation and architecture evaluation, which are currently in practice. We used these key architecture process activities as a set of independent variables in the

empirical investigation presented in this paper in order to construct the research model of our investigation.

2. Research models and hypotheses of the study

The main objectives of this empirical investigation are twofold. First, this study provides an opportunity to empirical investigate the association between the key architecture factors and software product line performance. Secondly, it studies the interrelationships of key architecture factors in managing and developing software product line architecture. Fig. 1 shows the theoretical model purposely designed for the empirical investigation of the association of software product line architecture process activities and software product line performance. The model examines the association of a number of independent variables arising from the literature survey of software product line engineering on the dependent variable of software product line performance of an organization. The main objective of this model is to investigate empirically the answer of the following research question:

RQ: What is the impact of architecture process activities on the overall performance of software product line?

There are six independent and one dependent variable of this model. The six independent variables are called "architecture factors" in the rest of this paper. They include domain engineering, requirements modeling, commonality management, variability management, architecture evaluation and architecture artifacts management. The dependent variable of this study is software product line performance of an organization. We measure the software product line performance of an organization for the past 3 years, with respect to cost and development time reductions, market growth, and financial strengths. The multiple linear regression equation of the model is depicted by Eq. (I)

$$\text{Software product line performance} = \beta_0 + \beta_1 f_1 + \beta_2 f_2 + \beta_3 f_3 + \beta_4 f_4 + \beta_5 f_5 + \beta_6 f_6 \quad \text{(I)}$$

where β_0, β_1, β_2, β_3, β_4, β_5, β_6 are coefficients, and f_1, f_2, f_3, f_4, f_5, f_6 are the six independent variables. In order to empirically investigate the research question (RQ) we hypothesize the following:

H1: Domain engineering activity has a positive impact on software product line performance.
H2: Requirements modeling is positively associated with the performance of software product line.
H3: Commonality management plays a positive role in software product line performance.
H4: Variability management is positively associated with software product line performance.
H5: Architecture evaluation is positively associated with software product line performance.
H6: The performance of software product line is positively associated with architecture artifacts management.

Fig. 2 illustrates the research model designed to investigate the interdependency of software product line architecture process activities. The model shown in Fig. 2 describes the directional association among various key architecture factors. The purpose of this research model is to find out the interrelationships of various key architecture factors of software product line. The main objective of this research model is to investigate empirically the answer of the following research question:

RQ-1: What is the impact of domain engineering activity on variability and commonality management of software product line architecture?
RQ-2: Does requirements modeling helps in managing and understanding variability and commonality in software product line architecture?

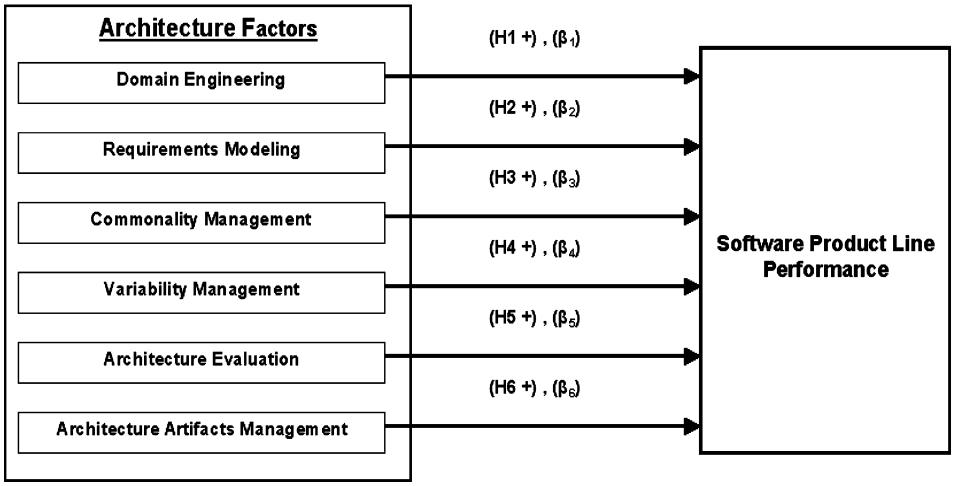

Fig. 1. Research model.

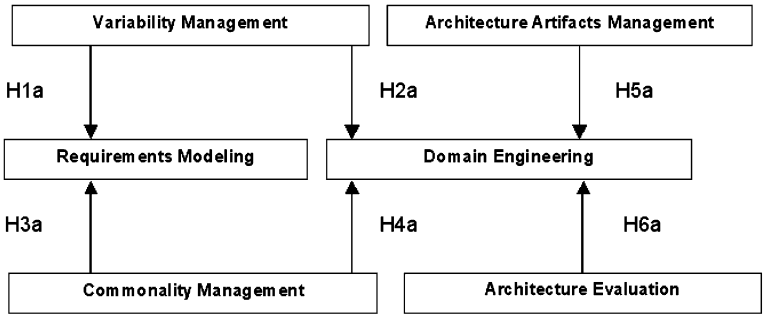

Fig. 2. Research model of inter-relationships of architecture factors.

RQ-3: Does domain engineering helps in managing product line architecture artifacts?

RQ-4: What is the association of architecture evaluation on the domain engineering?

In order to empirically investigate the research questions RQ-1 to RQ-4 we hypothesize the following:

H1a: Variability management is positively associated with requirements modeling.

H2a: Variability management is positively associated with domain engineering.

H3a: Commonality management has a positive association with requirements modeling.

H4a: Commonality management is positively associated with domain engineering.

H5a: Architecture artifacts management of software product line has positive relationship with domain engineering.

H6a: Architecture evaluation is positively associated with domain engineering.

3. Research methodology

The target population for this study was organizations, having been involved in using the software product line approach for more than 3 years. We approached organizations by sending them personalized emails and request them to participate in this empirical study. The organizations were requested to fill out the questionnaire purposely designed for this study. Ten organizations agreed to participate in this study with a mutual understanding of keeping the names of the organizations as well as individuals confidential. The participating organizations are involved in wide range of operations such as consumer electronics, telecommunication, avionics, automobiles, and information technology. The participating organizations are North American and European multinational companies. The organizations differed in size and range from medium to large-scale. We assume that the medium scale organization has number of employees around 2000–3000, whereas a large-scale organization has more then 3000 employees. It is important to note here that the size of the organization in terms of number of employees is based on total number of employees in the organization working in various departments. We requested the organizations under study to distribute the questionnaire within various departments, so that we could have many responses within same organization. The respondents, on average, had been associated with the organizations for the last 3 years. The minimum qualification of respondents was an undergraduate university degree and the maximum was a Ph.D. degree. We received a minimum of one and a maximum of six responses from each organization. The total respondents were 33 altogether.

3.1. Measuring instrument

The research model shown in Fig. 1, identified six independent variables termed as architecture factors in this study and one dependent variable of software product line performance. We collected data on the architecture factors and the perceived level of software product line performance using the questionnaire specifically designed for this study. The questionnaire presented in Appendix A was used to serve as a source of first contact in learning the extent to which the architecture factors were practiced within each organization dealing in the software product lines and their perceived level of organizational performance in software product line concept. The questionnaire required respondents to indicate the extent of their agreement or disagreement with statements using a five-point Likert scale. We used twenty-four separate items to measure the independent variables. We used four items to measure each organization's performance in the software product lines. Previous researches on software engineering, software architecture, software product lines, and software product line architecture were reviewed to ensure that a comprehensive list of measures were included in constructs of each independent variable. In order to measure the extent to which each of the six architecture factors were practiced in organizations dealing with software product lines; we used multi-item, five-points Likert scales that ranged from "strongly disagree" (1) to "strongly agree" (5) for all items associated with each variable. Four items for each independent variable were designed to collect measures on the extent to which the variable is practiced within each organization. The items for all six architecture factors are labeled sequentially in Appendix A and are numbered 1–

24. We measured the dependent variable, i.e. software product line performance for the past 3 years, with respect to cost and development time reductions, market growth, and financial strengths based on the multi-item, five-point Likert scale. The items were specifically designed for collecting measures for this variable and are labeled sequentially from 1 to 4 in Appendix A. All items shown in Appendix A are written specifically for this empirical investigation.

3.2. Reliability and validity analysis of measuring instrument

Conducting reliability and validity analysis, which has been widely used in empirical software engineering, can reduce the external threats to the findings of empirical investigations. Reliability refers to the reproducibility of a measurement, whereas validity refers to the agreement between the value of a measurement and its true value. We conducted reliability and validity analysis of the measuring instruments designed specifically for this empirical investigation by using most common approaches generally used in the empirical studies. The reliability of the multiple-item measurement scales of the six architecture factors was evaluated by using internal-consistency analysis. Internal-consistency analysis was performed using coefficient alpha [17]. Table 1 reports the reliability analysis, the coefficient alpha ranges from 0.60 to 0.88. Nunnally and Bernste [48] found that a reliability coefficient of 0.70 or higher for a measuring instrument is satisfactory. The other reliability literature such as [62] suggests that a reliability coefficient of 0.55 or higher is satisfactory, and Osterhof [50] concluded that 0.60 or higher is satisfactory. Therefore we determined that all variable items developed for this empirical investigation were reliable. We observed the content validity of the items included in each architecture factor, following the general recommendations of Cronbach [18] and Straub [56], by carrying out a comprehensive literature survey to include possible items in the variable scales. We also held discussions with the representatives of the organizations to finalize the proposed independent variables and items included in each variable which helps in managing face validity of the measuring instrument as well. We also conducted pilot tests, which led to modifications in the variable items, based on the suggestions of respondents, which improved the content and face validity.

Table 1
Coefficient alpha & principal component analysis of variables

Architecture factors	Item No.	Coefficient α	PCA eigen-value
Domain engineering	1–4	0.77	2.42
Requirements modeling	5–8	0.88	2.99
Commonality management	9–12	0.78	2.42
Variability management	13–16	0.79	2.47[a]
Architecture evaluation	17–20	0.66	2.23[a]
Architecture artifacts management	21–24	0.60	2.30

[a] Variable has more then one factors with eigen value >1.0.

Convergent validity, according to Campbell and Fiske [7], occurs when the scale items in a given construct move in the same direction (for reflective measures) and, thus, highly correlate. The principal component analysis [15] performed and reported for all six architecture factors in Table 1 provide a measure of convergent validity. We used eigen values [34] and scree plots [8] as reference points to observe the construct validity using principal component analysis. In this study, we used eigen value-one-criterion, also known as Kaiser criterion [35,55], which means any component having an eigen value greater then one was retained. Eigen values analysis revealed that four out of six variables completely formed a single factor, whereas in the case of variability management and architecture evaluation two components are formed. The eigen value for the second components are slightly higher then the threshold of 1.0. The scree plots clearly showed a cut-off at the first component. Therefore, the convergent validity can be regarded as sufficient. We used multiple regression analysis to determine the criterion validity of the six architecture factors and software product line performance. Architecture factors were used as predictor variables and software product line performance was used as a criterion variable. The multiple correlation coefficient observed, was 0.88. Cohen [14] concluded that a multiple correlation coefficient higher than 0.51 corresponds to a large effect size. Therefore, we observed the criterion validity of the variables to be sufficient. The measurements of reliability and validity analysis showed that the measurement procedures used in this study had the required level of psychometric properties.

3.3. Data analysis techniques

To analyze the research model and check the significance of the hypotheses H1–H6 and H1a–H6a, we used various statistical analysis techniques and initially divided the data analysis activity into three phases. Phase-I dealt with normal distribution tests and parametric statistics, whereas Phase-II dealt with non-parametric statistics. In order to reduce the threats to external validity due to low sample size we used both statistical approaches of parametric and non-parametric. We tested for the normal distribution of all the architecture factors using mean, standard deviation, kurtosis and skewness techniques, and found the values for all these tests to be within the acceptable range for the normal distribution with some exceptions. We made some modification to the data received from respondents before performing statistical analysis. Since all the six independent variables and the dependent variable's measuring instrument had multiple items, therefore we added their ratings to obtain a composite score for that measure before performing statistical analysis. The statistical analysis results reported in this paper are based on data received from all the respondents We conducted tests for hypotheses H1–H6 and H1a–H6a using parametric statistics, such as the Pearson correlation coefficient and one tailed *t*-test in Phase-I. In Phase-II of

non-parametric statistics, we conducted tests for hypotheses H1–H6 and H1a–H6a using Spearman correlation coefficient. Phase-III dealt with testing the hypotheses of the research models of this study using the technique of Partial Least Square (PLS). PLS technique helps when complexity, non-normal distribution, low theoretical information, and small sample size are issues [27,33]. Since small sample size was one of the major limitations in this study, therefore, we used PLS technique to increase the reliability of the results as well. One of the main reasons for small sample size is that software product line is a relatively young concept in the software industry and not many organizations are dealing in software product lines since the last 3 years. The statistical calculations were performed using Minitab® 14 software.

3.4. Low sample size issues and handling approach

In this study 10 organizations participated with a total number of respondents 33. The lower sample size in terms of number of organizations and respondents has a potential threat to the external validity of this study in order to generalize the outcomes. In this section we discuss why the sample size is low and how we handled this issue to reduce the threats to external validity. Following is the reason for the low sample size:

- The major reason behind small number of participating organizations is our initial criteria set of 3 years of experience in software product line development. There are not many organizations having the required level of experience in the business of software product line in particular due to relative young age of this concept. The reason behind choosing the 3 years experience in software product line, as a criteria set is the characteristics of long-term payback period of software product line. In order to enhance the external validity we intend to ensure that organizations have started assessing the benefits of software product line in terms of pay back or at least some potential benefits are apparent now. The number of respondents from organizations was beyond our control as we requested at the organizational level to distribute the survey and provide us feedback as well as all the participants of this study were volunteers.

We used many statistical techniques to ensure that with a low sample size we would observe reliable results to improve the external validity of the study. We took following measures to ensure the reliability of the results:

- We used both parametric (Pearson correlation) and non-parametric (Spearman correlation) statistical approaches to ensure the reliability of results. Since one technique is distribution dependent and other is not so we used both approaches to ensure the reliability of the results.

Table 2
Minimal sample size requirement for Pearson & Spearman correlation (90% & 80% power)

Correlation coefficient value	Power of 90%		Power of 80%	
	Pearson correlation	Spearman correlation	Pearson correlation	Spearman correlation
0.1	854	1013	618	733
0.2	212	250	154	183
0.3	93	107	68	79
0.4	51	62	38	46
0.5	32	39	24	30
0.6	21	26	16	20
0.7	15	19	12	15

- We also used Partial Least Square (PLS) technique to cross validate the statistical outcomes. Other studies [27,33] show that PLS provides more reliable results when low sample size is an issue.
- Values in Table 2 based on literature [14] provide information about minimal sample size requirements for Pearson and Spearman correlation coefficient at a power of 90% and 80%. We tested hypothesis using the Pearson and Spearman correlation coefficient. We observed a maximum 0.83 statistically significant correlation at $P < 0.05$, minimum 0.30 and average 0.65 (recorded in Table 3). According to Table 2 a sample size of 21 and 16 is required for Pearson correlation at a power of 90% and 80%, respectively. Whereas a sample size of 26 and 20 is required for Spearman correlation at a power of 90% and 80%, respectively, in case of an average correlation of 0.65 that we observed.

4. Data analysis and results

4.1. Hypotheses testing Phase-I

We examined the Pearson correlation coefficient and *t*-test between individual independent variables (architecture factors) and the dependent variable (software product line performance) of the research model shown in Fig. 1 in order to test hypotheses H1–H6, as well as the inter-relationships of different architecture factors shown in Fig. 2 to test hypotheses from H1a to H6a. The results of the statistical calculations for the Pearson correlation coefficient is reported in Table 3. The Pearson correlation coefficient between domain engineering and software product line performance was positive (0.68) at $P < 0.05$, and thus provided a justification to accept the H1 hypothesis. The hypothesis H2 was accepted based on the Pearson correlation coefficient (0.77) at $P < 0.05$ between requirements modeling and performance. The correlation coefficient of 0.81 at $P < 0.05$ was observed between the software product line performance and commonality management. The positive correlation coefficient of 0.83 at $P < 0.05$ meant that H4 was accepted. The hypotheses H5 (correlation: 0.02) was not found significant at $P < 0.05$, therefore the hypotheses H5 that deals with architecture evaluation and software product line perfor-

Table 3
Hypotheses testing using parametric and non-parametric correlation coefficients

Hypothesis	Research variables involved	Pearson coefficient	Spearman coefficient
H1	Domain engineering & software product line performance	0.68[a]	0.64[a]
H2	Requirements modeling & software product line performance	0.77[a]	0.75[a]
H3	Commonality management & software product line performance	0.81[a]	0.82[a]
H4	Variability management & software product line performance	0.83[a]	0.85[a]
H5	Architecture evaluation & software product line performance	0.02[b]	0.05[b]
H6	Architecture artifacts management & software product line performance	0.80[a]	0.81[a]
H1a	Variability management & requirements modeling	0.60[a]	0.62[a]
H2a	Variability management & domain engineering	0.64[a]	0.57[a]
H3a	Commonality management & requirements modeling	0.65[a]	0.66[a]
H4a	Commonality management & domain engineering	0.65[a]	0.67[a]
H5a	Architecture artifacts management & domain engineering	0.49[a]	0.43[a]
H6a	Architecture evaluation & domain engineering	0.36[a]	0.30[a]

[a] Significant at $P < 0.05$.
[b] Insignificant at $P > 0.05$.

mance is rejected. Hypothesis H6 was accepted after analyzing the Pearson correlation coefficient (0.80 at $P < 0.05$). The hypothesis H1a deals with variability management and requirements modeling and we observed a positive relationship (Person correlation coefficient: 0.60 at $P < 0.05$). Variability management has also a positive relationship with domain engineering thus allowing H2a to accept (Person correlation coefficient: 0.64 at $P < 0.05$). The hypothesis H3a was accepted based on the Pearson correlation coefficient (0.65) at $P < 0.05$, between commonality management modeling and requirements modeling. The Pearson correlation coefficient between commonality management and domain engineering was positive (0.65) at $P < 0.05$, and thus provided a justification to accept the H4a hypothesis. Architecture Artifacts management and domain engineering shows positive relationship (Person correlation coefficient: 0.49 at $P < 0.05$), hence H5a is accepted. The hypothesis H6a that deals with architecture evaluation and domain engineering is observed positive with Person correlation coefficient of 0.36 at $P < 0.05$. It was observed and is reported here that hypotheses H1, H2, H3, H4, and H6, of research model shown in Fig. 1, are found statistically significant and accepted whereas H5 is not supported therefore was rejected. Furthermore it was also observed and is reported here that hypotheses H1a, H2a, H3a, H4a, H5a and H6a are found statistically significant and accepted.

4.2. Hypotheses testing Phase-II

In Phase-II we conducted non-parametric statistics using Spearman correlation coefficient to test the hypotheses H1–H6 and H1a–H6a. Table 3 also reported the observation made in this testing phase. Hypothesis H1 was statistically significant at $P < 0.05$ with Spearman correlation coefficient of 0.64. A positive association is observed between requirements modeling and software product line performance (Spearman: 0.75 at $P < 0.05$). H3, which deals with commonality management and software product line performance, was accepted (Spearman: 0.82 at $P < 0.05$). The Spearman correlation of (0.85 at $P < 0.05$) is observed for H4. The hypothesis H5 between architecture evaluation and software product line performance was found statistically insignificant because the observed P-value was greater then 0.05. A positive Spearman correlation of 0.81 at $P < 0.05$ results in accepting H6. The Spearman correlation of (0.62 at $P < 0.05$) was observed between variability management and requirements modeling (H1a). H2a tests the relationship between variability management and domain engineering and found positive (Spearman correlation: 0.57 at $P < 0.05$). H3a deals with commonality management and requirements modeling and observed positive (Spearman correlation: 0.66 at $P < 0.05$). Domain engineering and commonality management (H4a) showed a positive association with Spearman correlation of 0.67 at $P < 0.05$. Hypothesis H5a (architecture artifacts management and domain engineering) was statistically significant at $P < 0.05$ with Spearman correlation coefficient of 0.43. H6a (architecture evaluation and domain engineering) was accepted based on Spearman correlation of 0.30 at $P < 0.05$. Hence, it was observed and is reported here that hypotheses H1, H2, H3, H4, and H6, are found statistically significant and accepted whereas H5 is not supported thus rejected. Further more hypotheses to test the interrelationships of architecture factors (H1a–H6a) are found statistically significant and accepted.

4.3. Hypotheses testing Phase-III

In Phase-III of hypotheses testing, we used the PLS technique to overcome some of the associated limitations and to cross validate with the results observed using approach of Phase-I and Phase-II. We tested the hypothesized relationships, i.e. H1–H6, as well as H1a–H6a by examining their direction and significance. In PLS testing of H1–H6 we placed software product line performance as response variable and individual architecture factor as predicate. In order to test hypotheses H1a–H6a we placed one variable as predicator and other as response to test their association. Table 4 reports the result of structural tests of the hypotheses. It contains detailed observed values

Table 4
Hypotheses testing using partial least square regression

Hypothesis	Research variables involved	Path coefficient	R^2	F-Ratio
H1	Domain engineering & software product line performance	0.71	0.46	26.92[a]
H2	Requirements modeling & software product line performance	0.53	0.59	46.22[a]
H3	Commonality management & software product line performance	0.63	0.66	61.77[a]
H4	Variability management & software product line performance	0.74	0.69	70.71[a]
H5	Architecture evaluation & software product line performance	0.02	0.005	0.02[b]
H6	Architecture artifacts management & software product line performance	0.95	0.65	57.90[a]
H1a	Variability management & requirements modeling	0.46	0.36	18.13[a]
H2a	Variability management & domain engineering	0.76	0.42	22.60[a]
H3a	Commonality management & requirements modeling	0.57	0.42	23.17[a]
H4a	Commonality management & domain engineering	0.88	0.43	23.66[a]
H5a	Architecture artifacts management & domain engineering	0.44	0.24	10.29[a]
H6a	Architecture evaluation & domain engineering	0.40	0.12	4.61[a]

[a] Significant at $P < 0.01$.
[b] Insignificant at $P > 0.05$.

of path coefficient, $R^2$and F-ratio. The path coefficient of domain engineering was found 0.71, R^2: 0.46 and F-ratio (26.92) was significant at $P < 0.01$. Requirements modeling: 0.53, R^2: 0.59, F-ratio: 46.22 at $P < 0.01$ have the same direction as proposed. Commonality management (Path coefficient: 0.63, R^2: 0.66, F-ratio: 61.77 at $P < 0.01$) also has the same direction as proposed in H3. Variability management (Path coefficient: 0.74, R^2: 0.69, F-ratio: 70.71 at $P < 0.01$) is also found in accordance with H4. Architecture evaluation has path coefficient of 0.02 at a very low R^2 of 0.005 and F-ratio of 0.02 was not found significant at $P < 0.05$. The path coefficient of architecture artifacts management was found 0.95, R^2: 0.65 and F-ratio (57.90) was significant at $P < 0.01$. H1a deals with variability management and requirements modeling path coefficient of the relationship was found 0.46, R^2: 0.36 and F-ratio (18.13) was significant at $P < 0.01$. The path coefficient of the hypothesis H2a between variability management and domain engineering was observed 0.76, R^2: 0.42 and F-ratio (22.60) was significant at $P < 0.01$. The association of commonality management and requirements modeling (H3a) has the same direction as proposed (path coefficient: 0.57, R^2: 0.42, F-ratio: 23.17 at $P < 0.01$). H4a is accepted based on path coefficient of 0.88, R^2: 0.43, F-ratio: 23.66 at $P < 0.01$. Architecture artifacts management and domain engineering (H5a) showed a positive relationship with a path coefficient of 0.44, R^2: 0.24, F-ratio: 10.29 at $P < 0.01$. The hypothesis H6a between architecture evaluation and domain engineering was found statistically significant (path coefficient: 0.40, R^2: 0.12, F-ratio: 4.61 at $P < 0.01$). All in all hypotheses H1, H2, H3, H4, and H6 showed significant at $P < 0.01$ with a positive path coefficient and are in the same direction as proposed. The hypotheses H5 that deals with architecture evaluation and software product line performance was not found statistically significant at $P < 0.05$. The PLS analysis further showed that the hypotheses H1a–H6a are found significant at $P < 0.01$ with a positive path coefficient and are in the same direction as proposed.

4.4. Testing of research model

The linear regression equation of the research model is illustrated by Eq. (I). The purpose of model testing is to provide empirical evidence that architecture factors play a considerable role in software product line performance. The testing process consists of conducting regression analysis and reporting the values of the model coefficients and their direction of association. We placed software product line performance as response variable and individual architecture factor as predicators. The analysis also reports the results of two-tailed t-tests conducted and their statistical significance. Table 5 reports the regression analysis of the research model. The path coefficient of five out of six variables: domain engineering, requirements modeling, commonality management, variability management, and architecture artifacts management are found positive and their t-statistics is also observed statistically significant at either $P < 0.01$ or $P < 0.05$. The path coefficient of architecture evaluation is found negative. Negative t-statistics and $P > 0.05$ make architecture evaluation statistically insignificant in this research model. The adjusted R^2 of overall

Table 5
Linear regression analysis of research model

Model coefficient name	Model coefficient	Coefficient value	t-Value
Domain engineering	β_1	0.22	2.26[b]
Requirements modeling	β_2	0.19	3.02[a]
Commonality management	β_3	0.16	2.06[b]
Variability management	β_4	0.19	2.08[b]
Architecture evaluation	β_5	−0.08	−1.08[c]
Architecture artifacts management	β_6	0.26	2.37[b]
Constant	β_0	0.46	0.32[c]
R^2	0.90	Adjusted R^2	0.88
F-Ratio	42.09[a]		

[a] Significant at $P < 0.01$.
[b] Significant at $P < 0.05$.
[c] Insignificant at $P > 0.05$.

research model was observed 0.88 with a F-ratio of 42.09 significant at $P < 0.01$.

5. Sensitivity analysis of the study

The challenges of empirical studies include reducing threats to external validity and increasing reliability because empirical investigations are subject to a number of limitations, which may contribute in threats to external validity and reliability. Results from an empirical study depend on data, the statistical model and the statistical techniques used and sensitivity analysis is defined as the investigation of how research model misspecification and anomalous data points influence results [49]. According Kitchenham et al. [39], in empirical investigations it is important to perform a sensitivity analysis to understand how individual data points or clusters of data relate to the behavior of the whole collection. Saltelli et al. [52] define sensitivity analysis as the study of how the variation in the output of a model can be apportioned, qualitatively or quantitatively, among model inputs. The sensitivity analysis allows the researchers to understand how the research model behaves on changing inputs and it further supports the empirical investigations in terms of reliability and validity. In this section we reported two sensitivity analysis tests. First test deals with inter-rater agreement because we have varying number of respondents within same organization. Second test addresses the sensitivity analysis of the overall research model shown in Fig. 1.

5.1. Inter-rater agreement analysis

We received more then one number of responses from seven organizations out of ten participating organizations. Varying number of respondents within an organization may have conflicting opinions about the performance of software product line with in the same organization, moreover the respondents belongs to various departments within same organizations as well, so there is a need to perform inter-rater agreement analysis which would provide information about the extent of agreement among the raters within one organization. Inter-rater agreement corresponds to reproducibility in the evaluation of the same process according to the same evaluation specification [43]. According to El Emam [23] the inter-rater agreement is concerned with the extent of agreement in the ratings given by independent assessors to the same software engineering practices. The Kendall coefficient of concordance (W) [64] is often preferred to evaluate inter-rater agreement in comparison to some other methods such as Cohen's Kappa [13] in case of ordinal data. "W" is an index of the divergence of the actual agreement shown in the data from the possible perfect agreement. In order to ensure the reliability and validity of this empirical investigation, we conducted and reported the inter-rater agreement analysis using Kendall's and Kappa statistics. The Table 5 reports the Kendall and Kappa statistics of seven organizations participated in this study with more then one respondents. Values of Kendall's W and Fliess Kappa coefficient can range from 0 to 1, with 0 indicating perfect disagreement, and 1 indicating perfect agreement [42]. The values of the Kendall coefficient of concordance (W) in Table 6 range from 0.38 to 0.75 with an average value of 0.58, whereas Fliess Kappa coefficient values range from 0.11 to 0.51. The inter-rater agreement analysis shown in Table 6 concludes not a perfect agreement which rarely happens in case of empirical studies involving several respondents but leads to a conclusion of substantial agreement.

Table 6
Inter-rater agreement analysis

Organization	Kendall statistics		Kappa statistics	
	Kendall's coefficient of concordance (W)	χ^2	Fleiss kappa coefficient	Z
A	0.75	69.70[a]	0.51	8.42[a]
B	0.58	66.70[a]	0.31	6.39[a]
C	0.71	65.69[a]	0.45	6.70[a]
D	0.47	65.97[a]	0.18	4.06[a]
E	0.50	35.03[a]	0.11	1.15[b]
F	0.38	53.18[a]	0.22	4.45[a]
G	0.66	30.70[b]	0.24	1.25[c]

[a] Significant at $P < 0.01$.
[b] Significant at $P < 0.05$.
[c] Insignificant at $P > 0.10$.

5.2. Sensitivity analysis of the research model

The research model purposely designed for this empirical study shown in Fig. 1 consists of six-independent variables and one dependent variable. The results of empirical investigation reported in Section 4 show that five out of these six independent variables are positively associated with the dependent variable of software product line performance. This empirical investigation did not provide significant statistical support for the positive association of architecture evaluation and the performance of software product lines. The purpose of sensitivity analysis of research model is to study the impact of each independent variable on the overall output of the model especially to study the impact of architecture evaluation on the overall output of the model because it has not been supported by the study. We used Fourier Amplitude Sensitivity Test (FAST) and Sobol methods to conduct and report sensitivity analysis of the research model of this study. FAST method is commonly used to estimate the ratio of the contribution of each input to the output variance with respect to the total variance of the output as the first order sensitivity index. FAST can identify the contribution of individual inputs to the expected value of the output variance [19]. FAST does not assume a specific functional relationship such as linear or monotonic in the model structure, and thus works for both monotonic and non-monotonic models [52]. Sobol method [54] apportions the output variance among individual inputs and their interactions. The

Table 7
Sensitivity analysis of the research model

Architecture factors	FAST sensitivity index (%)	Soboll sensitivity index (%)
Domain engineering	16	15
Requirements modeling	25	26
Commonality management	15	16
Variability management	15	10
Architecture evaluation	1	1
Architecture artifacts management	16	15

method of Sobol can cope with both non-linear and non-monotonic models, and provides a quantitative ranking of inputs [9]. Sobol's method provides insight with respect to the main effect, interaction effect, and total effect of each input with respect to the output of interest. The main effect of each input represents the fractional unique linear contribution of the input to the output variance. The sensitivity analysis of the research model is reported in Table 7. It illustrates that all the five independent variables, which are positively associated with the dependent variable of the research model, contribute significantly to the overall output of the model. Architecture evaluation, which is not significantly associated with the performance of software product line, has very low contribution (FAST: 1%, Sobol: 1%) in the overall output of the model. The sensitivity evaluation calculations were performed using SimLab 2.2 software.

6. Discussion

This research enables organizations to understand the effectiveness of the relationships and inter-dependency of key architecture process activities and software product line. This study provides an opportunity to empirically investigate the association between the key architecture process activities and software product line performance as well as their inter-dependency. The results provide the first empirical evidence that key architecture process activities play a critical role in the development of software product line within an organization. The organization in the business of software product line has to deal with multiple key software product line architecture process activities in addition to their efforts in software development.

Domain engineering has a pivotal role in the process of software product line. This study finds a positive association between domain engineering and software product line performance. The inception phase of software product line starts with conducting a comprehensive domain engineering in defining and narrowing down the scope of product line, which identifies the characteristics of the product line and the products that comprise the product line. The product line engineering envisages the domain engineering into set of three activities: domain analysis, domain design and domain implementation. Domain analysis concentrates on understanding the domain and providing a foundation to domain design, which is an early sketch of the architecture of product line. Domain analysis not only defines the boundaries of the software product line scope but also helps in performing the commonality and variability analysis for the product line. Domain implementation further helps in developing the core architecture of software product line by specifying components and their inter-connection. The activities of domain engineering invariably provide helps in carrying out commonality and variability analysis. The domain engineering helps in defining the common and variable parts of the software product line requirements, thus explicitly identifying the commonality and variability of the envision products. This study also confirms this finding and reports the positive association of domain engineering with commonality and variability management. The software product line requires a strong coordination among domain engineering and application engineering. The domain engineering helps in establishing an infrastructure for software product line and the application engineering uses the infrastructure and develops products using core assets.

Requirements modeling provides us facility to model the requirements graphically so that requirements can be easily understand by various stakeholders Requirements modeling help in understanding the requirements of the products and further elaborates the functionalities and tradeoffs. Software product line needs to elaborate the requirements at two levels: product line level and individual product level. The product line level requirements envisage the commonality among products whereas individual product level requirements represent the variability. We found a positive impact of requirements modeling on the performance of software product line in this empirical investigation. We also observed a positive association of requirements modeling with commonality and variability management. Requirements modeling in the context of software product line architecture helps in identifying and specifying the extension points commonly known as variation points in software product line literature. It decomposes and specifies the architecture into set of features with their dependency. Requirements models translates the requirements of the targeted market segment and specify the implementation views of the business case. Much of the work on requirements modeling for software product line has concentrated on establishing an extension in the current available modeling techniques like UML and feature diagrams.

Product requirements in software product line are composed of a constant and a variable part. The constant part comes from product line requirements and deals with features common to all the products belonging to the family. The variable part represents those functionalities that can be changed to differentiate one product from another. This causes the significance of commonality and variability management in software product line. Commonality among products of a software product line is an essential and integral characteristic of product line approach that paves a way to maximize reusability. The products share the common architecture and they are developed out of

common core assets. In this empirical investigation we found a positive association between commonality management and the performance of software product line. This study also found a positive association of domain engineering and requirements modeling with the commonality management. The commonality management takes much if its' input from domain engineering and those inputs are further elaborated and clearly specified using requirements modeling. The extent of commonality among products is a design decision based on business case engineering and targeted market segment. In order to maximize the reusability of software assets it is generally recommended to have as much commonality as possible.

Variability among products of a software product line is necessary because it makes them a separate business entity. The products from the software product line may vary from each other's in quality, reliability, functionality, performance and so on, but as they share the common architecture so the variation should not be that much high so that they become out from the scope of the product line. Those variations must be handled systematically to accommodate changes in various versions of the product. The objective of variability management is to identify, specify and document variability among products in the applications of product line. Software product line architecture represents variability by specifying the variation points, which can be exploited at application engineering level by accommodating the design decisions based on the business case. The variability in products has influence from internal and external factors. The internal factors have their roots in refining the architecture whereas external factors accommodate the market and customers expectations. The introduction of variable features in a product from a software product line is a strategic decision based on market segment. The findings of this empirical investigation confirm a positive relationship between variability management and software product line performance. This study also observed that a better variability management is also dependent on the activities of domain engineering and requirements modeling. The introduction of variable features in the successive products out of product line also provides a justification for setting up a product line in the organization as well because it helps in attracting new customer and retaining the current one. Fitting the component into the product without tailoring it is the easiest task, but some time we need to make certain changes in the component to meet the requirements for a particular product. Every component present in the core assets must clearly define the variability mechanism to be used in order to tailor them for reusing. The significance of commonality and variability management in software product line architecture and the overall performance of the software product line require tool support, which needs the attention of researchers.

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

The findings of this empirical investigation did not statistically provide a significant support to the positive association of architecture evaluation and software product line performance. Although the direction of association between the performance of software product line and architecture evaluation was found negative, the significant statistical level of confidence did not support that result. Therefore it is concluded that this study is not able to find answer about the association and impact of architecture evaluation and software product line performance. Although this empirical investigation is unable to find an association of architecture evaluation and software product line performance, the theoretical foundations of this concept foresee a strong relationship among them.

6.1. Limitations of the study and threats to external validity

The empirical investigations are subject to certain limitations, although we did and report number of measures to reduce the threats to external validity and increasing the reliability of the study, there are still some limitations in this study. The first limitation is the selection and partic-

ipating independent variables of the research model. We used six independent variables to relate with the dependent variable of software product line performance. There may be other contributing factors that influence the performance of software product lines in addition to these six, but we kept the scope of this study within architecture process activities. Some other contributing factors to performance of software product lines, such as: organization size, economic, experience in software development and political conditions were not considered in this study. The second limitation is bias, which is a coherent limitation of almost all empirical studies. Although we used multiple respondents within the same organization to reduce bias, bias still is a core issue in decision-making. We asked the respondents to consult major sources of data at their organization, i.e., documents, plans, models, and actors before responding to a particular item in order to reduce the human tendency to over- or under-estimate when filling in questionnaires. The items were designed using accepted psychometric principles, but the measurement is still largely based on the subjective assessment of an individual. The third notable limitation of this study is small sample size. Software product line is a relatively young concept in software development, and not many of the organizations in the software industry have institutionalized and launched this concept, so collecting data from the software industry was a limitation. The lower sample size in terms of number of organizations and respondents has a potential threat to the external validity of this study. One major reason behind small number of participating organizations is that there are not many organizations involve in the business of software product line in particular due to relative young age of this concept. The number of respondents from organizations was beyond our control as we requested at the organizational level to distribute the survey and provide us feedback. Besides its general and specific limitations, this work contributes significantly in the area of software product lines and helps to understand the architecture dimension of software product lines.

6.2. Ethical issues and software empirical studies

Surveys, experiments, metrics, case studies, and field studies are examples of empirical methods used to investigate both software engineering processes and products [53]. The increased popularity of empirical methodology in software engineering has also raised concerns on the ethical issues. This paper reported an empirical investigation to find out certain key architecture process activities that have an impact on the performance of software product line in an organization. We followed the recommended ethical principles to ensure that the empirical investigation conducted and reported in this paper would not violate any form of recommended experimental ethics. The primary ethical principle in human subject research is that of full informed consent on the part of the subject to participate in the research project [26,37]. We fully informed the participating persons in this empirical investigation about the nature and objectives of this study. Moreover the information we acquired from respondents does not reflect the personal information of the individual rather they were requested to provide us information of their own judgment about up to what extent a key architecture process activity is effectively carrying out in their organizations. Therefore the information we collected was at organizational level not at individual personal level. According to Singer and Vinson [55] ethics do not fully agree on the necessary components of informed consent, but it is clear that it must contain at least some of the elements such as: disclosure, comprehension and competence, voluntariness and the right to withdraw from the experiment. We ensured the participants in writing that the survey conducted for this empirical investigation is a part of a Ph.D. research and neither the identity of an individual nor of an organization will be disclosed in the Ph.D. thesis or in any research publications. We send the questionnaires and received responses from the experts in the area of software product lines in order to ensure the competence of the respondents. All the participants of this study were volunteers and no compensation in any form was offered or paid. We also mentioned to the respondents that if for any reason they do not want to answer any question, please leave it blank. We believe that the study conducted and reported in this paper followed the recommended ethical principles.

7. Conclusion and future work

This work facilitates a better understanding of the architecture dimension of software product lines. Our first objective was to empirically investigate the effect of some key architecture process activities in the performance of software product lines thus finding answer to the research question (RQ) put forward. Results of this empirical investigation demonstrate that better architecture process activities contribute enhanced software product line performance. The results strongly support the hypotheses that domain engineering, requirements modeling, commonality management, variability management and architecture artifacts management are positively associated with the performance of software product lines in an organization. We did not find any significant statistical support for architecture evaluation in the better software product line performance and were not able to find answer about the association and impact of architecture evaluation and software product line performance.

The second objective of this study was to find answers of the research questions RQ-1, RQ-2, RQ-3 and RQ-4, which were designed to investigate the inter-dependency of architecture process activities.

- The findings of this empirical investigation support that domain engineering is positively associated with commonality and variability management (RQ-1).

- Requirements modeling further increase the understanding of commonality and variability management in products (RQ-2).
- The architecture artifacts management is positively associated with the domain engineering activity (RQ-3).
- Domain engineering further facilitates the architecture evaluation methodology (RQ-4).

Overall the second objective of this research enables organizations to understand the effectiveness of the interrelationships of architecture process activities and software product lines. The work conducted and reported in this paper is a first of its kind in the area of software product lines. This research reinforces current perceptions about the significance of architecture process activities and their involvement and impact in successful software product line development. Currently, we are working on developing a *Process Maturity Model* for process assessment of software product lines. This work has provided the empirical justification to include these architecture key process activities in evaluating the architecture dimension of software product line process maturity.

Appendix A. Architecture factors (Measuring instrument)

Domain engineering

1. The organization has adequate knowledge and resources about the domain of software product line.
2. Roles and responsibilities of individuals and groups are well defined and documented in the domain-engineering unit of the organization.
3. Domain requirements of software product line are clearly identified, stated and documented.
4. The domain analysis helps in setting up scope of software product line, which covers the domain of the application.

Requirements modeling

5. The organization uses a notation language to represent software product line requirements.
6. The organization develops requirements models for the better understanding of the software product line architecture, which explicitly shows the architectural structure.
7. The organization prepares requirements models, which help to visualize the interconnection of various sub-units of the software product line architecture by identifying candidate components, and their interconnection.
8. The requirements models are regularly reviewed, updated as needed and communicated to respective personnel in development.

Commonality management

9. The domain engineering activities in the organization identifies commonality among a set of envisioned product line applications.
10. The management encourages as much commonality as possible and developers concentrate more on product specific issues rather then issues common to all products.
11. Software product line requirements clearly identify, model and document commonality in products.
12. The commonality management allows us to maximize reuse in the organization.

Variability management

13. The domain engineering activities in the organization identifies variability among a set of envisioned product line applications.
14. Variability in products is within the scope of the software product line and design decisions of variability are influenced by market requirements and customers' expectations.
15. Requirements models clearly illustrate variability in products by showing variation points explicitly.
16. The variability in products help in retaining current customers and have a tendency to attract new customers.

Architecture evaluation

17. The organization has clear guidelines and well-documented methodology to evaluate the software product line architecture.
18. The simulations and prototyping activities are used to analyze the structure and interconnection of components.
19. The quality and functional attributes to evaluate the software product line architecture are explicitly defined.
20. The organization has explicitly defined specific qualitative metrics to measure the performance of the software product line architecture.

Architecture artifacts management

21. Architecture significant requirements are identified, elaborated and well documented.
22. We are using architecture description language to describe and document architectural structure and textures.
23. The architecture artifacts are regularly reviewed, updated if necessary and communicated to the developers.
24. The components description, interface requirements, interconnection hierarchy, variation mechanism are explicitly documented and traceable.

Software product line performance

1. Over the past 3 years, the organization is able to reduce cost, product defects and development time of software products.
2. The sales of the organization have steadily increased over the past 3 years and the organization is able to attract new customers and launch new products.
3. Software product line is playing a significant role in achieving the business goals of the organization.
4. Financial analysis shows a progressive growth over the last 3 years due to software product lines.

References

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

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

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

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

[5] J. Bosch, Design and Use of Software Architectures: Adopting and Evolving a Product-line Approach, Addison Wesley, 2000.

[6] G. Buckle, P.C. Clements, J.D. McGregor, D. Muthig, K. Schmid, Calculating ROI for software product lines, IEEE Software 21 (3) (2004) 23–31.

[7] D.T. Campbell, D.W. Fiske, Convergent and discriminant validation by the multi-trait multi-method matrix, Psychological Bulletin 56 (2) (1959) 81–105.

[8] R.B. Cattell, The scree test for the number of factors, Multivariate Behavioral Research 1 (1966) 245–276.

[9] K. Chan, S. Tarantola, A. Saltelli, I.M. Sobol, Variance-based Methods in Sensitivity Analysis, Wiley, New York, 2000.

[10] P.C. Clements, L.G. Jones, L.M. Northrop, J.D. McGregor, Project management in a software product line organization, IEEE Software 22 (5) (2005) 54–62.

[11] P.C. Clements, On the importance of product line scope, in: Proceedings of the 4th International Workshop on Software Product Family Engineering, 2001, pp. 69–77.

[12] P.C. Clements, L.M. Northrop, Software Product Lines Practices and Pattern, Addison Wesley, 2002.

[13] J. Cohen, A coefficient of agreement for nominal scales, Educational and Psychological Measurement 20 (1960) 37–46.

[14] J. Cohen, Statistical Power Analysis for the Behavioral Sciences, second ed., Lawrence Erlbaum Associates, Inc., Publishers, Hillsdale, NJ, 1988.

[15] A.L. Comrey, H.B. Lee, A First Course on Factor Analysis, second ed., Lawrence Erlbaum Associates, Inc., Publishers, Hillsdale, 1992.

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

[17] L.J. Cronbach, Coefficient alpha and the internal consistency of tests, Psychometrica 16 (1951) 297–334.

[18] L.J. Cronbach, Test validation, Educational Measurement (1971) 443–507.

[19] R.I. Cukier, H.B. Levine, K.E. Shuler, Nonlinear sensitivity analysis of multi-parameter model systems, Journal of Computational Physics 26 (1) (1978) 1–42.

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

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

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

[23] K. El Emam, Benchmarking kappa: inter-rater agreement in software process assessments, Empirical Software Engineering 4 (2) (1999) 113–133.

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

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

[26] R.R. Faden, T.L. Beauchamp, N.M.P. King, A History and Theory of Informed Consent, Oxford University Press, 1986.

[27] C. Fornell, F.L. Bookstein, Two structural equation models: LISREL and PLS applied to consumer exit voice theory, Journal of Marketing Research 19 (1982) 440–452.

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

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

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

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

[32] M. Jazayeri, A. Ran, F. van der Linden, Software Architecture for Product Families: Principles and Practice, Addison Wesley, 2000.

[33] K. Joreskog, H. Wold, Systems Under Indirect Observation: Causality, Structure and Prediction, North Holland, The Netherlands, 1982.

[34] H.F. Kaiser, A second generation little jiffy, Psychometrika 35 (1970) 401–417.

[35] H.F. Kaiser, The application of electronic computers to factor analysis, Educational and Psychological Measurement 20 (1960) 141–151.

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

[37] J. Katz, Experimentation with Human Beings, Russell Sage Foundation, New York, 1972.

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

[39] B.A. Kitchenham, S.L. Pfleeger, L.M. Pickard, P.W. Jones, D.C. Hoaglin, K. El Emam, J. Rosenberg, Preliminary guidelines for empirical research in software engineering, IEEE Transactions on Software Engineering 28 (8) (2002) 721–734.

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

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

[42] J. Landis, G.G. Koch, The measurement of observer agreement for categorical data, Biometrics 33 (1977) 159–174.

[43] H.Y. Lee, H.W. Jung, C.S. Chung; J.M. Lee, K.W. Lee, H.J. Jeong, Analysis of inter-rater agreement in ISO/IEC 15504-based software process assessment, in: Proceedings of the 2nd Asia-Pacific Conference on Quality Software, 2001, pp. 341–348.
[44] R.R. Macala, L.D. Stuckey Jr., D.C. Gross, Managing domain-specific, product-line development, IEEE Software 13 (3) (1996) 57–67.
[45] M. Matinlassi, Comparison of software product line architecture design methods: COPA, FAST, FORM, KobrA and QADA, in: Proceedings of the 26th International Conference on Software Engineering, 2004, pp.127–136.
[46] K. Mika, M. Tommi, Assessing systems adaptability to a product family, Journal of Systems Architecture 50 (2004) 383–392.
[47] E. Niemelä, M. Matinlassi, A. Taulavuori, Practical evaluation of software product family architectures, in: Proceedings of the 3rd International Conference on Software Product Lines, 2004, pp. 130–145.
[48] J.C. Nunnally, I.A. Bernste, Psychometric Theory, third ed., McGraw Hill, New York, 1994.
[49] H. Nyqusit, Sensitivity analysis of empirical studies, Journal of Official Statistics 8 (2) (1992) 167–182.
[50] A. Osterhof, Classroom Applications of Educational Measurement., Prentice Hall, NJ, 2001.
[51] B.J. Pronk, An interface-based platform approach, in: Proceedings of the 1st Software Product Lines Conference, 2000, pp. 331–352.
[52] A. Saltelli, K. Chan, M. Scott, Sensitivity Analysis, Probability and Statistics Series, Wiley, NY, 2000.
[53] I.M. Sobol, Sensitivity estimates for nonlinear mathematical models, Mathematical Modeling Computers 1 (4) (1993) 407–414.
[54] J. Stevens, Applied Multivariate Statistics for the Social Sciences, Lawrence Erlbaum Associates, Inc., Publishers, Hillsdale, NJ, 1986.
[55] J. Singer, N.G. Vinson, Ethical issues in empirical studies of software engineering, IEEE Transactions on Software Engineering 28 (12) (2002) 1171–1180.
[56] D.W. Straub, Validating instruments in MIS research, MIS Quarterly 13 (2) (1989) 147–169.
[57] J.M. Thompson, M.P.E. Heimdahl, Structuring product family requirements for *n*-dimensional and hierarchical product lines, Requirements Engineering Journal 8 (1) (2003) 42–54.
[58] A. van der Hoek, E. Dincel, N. Medvidovic, Using service utilization metrics to assess the structure of product line architectures, in: Proceedings of the 9th International Software Metrics Symposium, 2003, pp. 298–308.
[59] R. van Ommering, Software reuse in product populations, IEEE Transactions on Software Engineering 31 (7) (2005) 537–550.
[60] F. van der Linden, Software product families in Europe: the Esaps & Café projects, IEEE Software 19 (4) (2002) 41–49.
[61] F. van der Linden, J. Bosch, E. Kamsties, K. Känsälä, H. Obbink, Software product family evaluation, in: Proceedings of the 3rd International Conference on Software Product Lines, 2004, pp. 110–129.
[62] A.H. van de Ven, D.L. Ferry, Measuring and Assessing Organizations, Wiley, NY, 1980.
[63] M. Verlage, T. Kiesgen, Five years of product line engineering in a small company, in: Proceedings of the 27th International Conference on Software Engineering, 2005, pp. 534–543.
[64] A. von Eye, E.Y. Mun, Analyzing Rater Agreement Manifest Variable Methods, LEA Publishers, London, 2005.
[65] D.M. Weiss, C.T.R. Lai, Software Product Line Engineering: A Family Based Software Development Process, Addison Wesley, 1999.
[66] H. Zhang, S. Jarzabek, B. Yang, Quality prediction and assessment for product lines, in: Proceedings of the 15th International Conference on Advanced Information Systems Engineering, 2003, pp. 681–695.
[67] H. Zuo, M. Mannion, D. Sellier, R. Foley, An extension of problem frame notation for software product lines, in: Proceedings of the 12th Asia Pacific Conference on Software Engineering, 2005, pp. 499–505.