Analysis and Comparison of Software Product Line Frameworks

Alireza Olyai*, Reza Rezaei
Shahid Beheshti University, Tehran, Iran.

* Corresponding author. Tel.:00989111131473; email: salireza.olyaee@gmail.com
Manuscript submitted January 9, 2015; accepted June 24, 2015.
doi: 10.17706/jsw.10.8.991-1001

Abstract: Software product line architectures have been much attention in the software research community in past years. Although many methods and frameworks have been used to create software product lines, but there are still many issues and challenges in this field. This paper describes the issues and challenges surrounding software product lines, then introduced four software product line frameworks. Finally, these frameworks have been compared based on the issues and challenges such that, each framework is improved which of the issues and challenges.

Key words: Challenges of product lines, software product lines, software product line frameworks, software architecture.

1. Introduction

The concept of software product line architecture back to years ago, when Parnas said that" we consider a set of programs to be a program family if they have so much in common that it pays to study their common aspects before looking at the aspects that differentiate them" [1]. Nowadays, software product line designed as an important issue in the domain of software development is considered by experts in software engineering. Software product line, defined as a set of software-intensive systems (this means that the software plays a critical role in their) which has a set of common features and managed, which satisfy the specific needs of the market and that are developed from a common set of core assets in a prescribed way [2], [3]. In fact software product line vision is a set of reusable assets that include the basic architecture and common elements and may be adjusted [2]. Software product line also includes designs and documents, user guides, project management artifacts such as budgets and schedules, and software test plans and test data [2]. Another definition of software product lines, in [4] is: "a software product line is a software system aimed at producing a set of software products by reusing a common set of features, or core asset, that are shared by these products". The benefits of the idea of software product line are reducing costs and improve time-to-market. The reason is that companies have turned to the use of software product line design. Fig. 1 shows the economic benefits of software product line. Software architecture represents a large investment in time and cost [2]. So it is natural to want to maximize the return on this investment by re-using architecture across other systems [2]. Organizations which have a maturity of architecture tend to their architecture, considered as a valuable asset, and as the brainchild of creative and look for ways in which their assets have a key role in the production of high-income and low-cost products [2]. Both are possible

with architecture re-use. When an organization is producing similar systems and using the same architecture, the organization obtains a large gain that includes the reduced cost of construction and reduced time to market [2]. Generally, the development of product line requirements an architectural approach in which the core architecture capture both the commonalities shared by all the products and the variability of individual products the so–called variation points [2], [5]. So, we can say a software product line consists of a family of software systems that have some common functionality and some variable functionality [6].

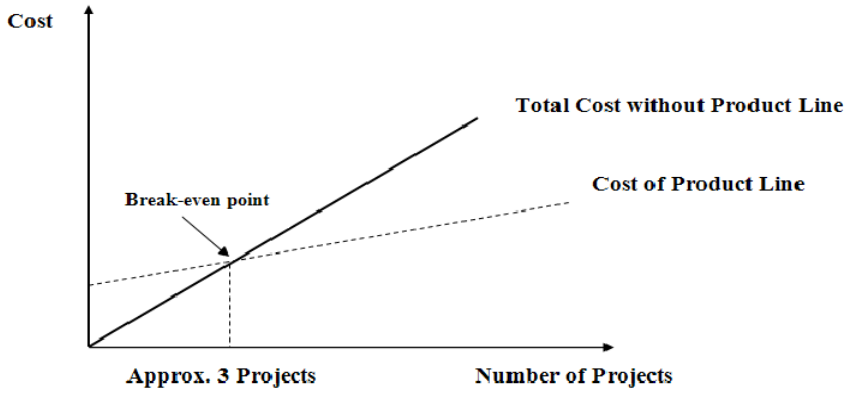

Fig. 1. Economics of software product line engineering similar to [7].

The reuse of software artifacts is a key factor in software product line, so it is important to build environments that support such practice, as well as a strong architecture related to it [8].Improving variability in a system implies making it easier to do certain kinds of changes [9].In variability a variation point represents a delayed design decision [10]. Managing the differences between products is a key success factor in software product families [11]. To design software product line architecture is proposed various methods or frameworks, but still these frameworks have not resolved all the issues and challenges associated with software product lines. Some of basic frameworks for software product line that have their roots in other frameworks could be COPA or component-oriented platform architecting is a software product line framework that is component-oriented. The goal of the COPA framework is the alignment between business, architecture, process and organization [12]. FAST or Family-Oriented Abstraction, Specification and Translation is a software development process is focused on the construction of product families [12]. In FAST framework divides the process of a product line into three sections, including the domain qualification, domain engineering and application engineering [13]. FORM or feature-oriented reuse method is a feature oriented method that, by analyzing the features of the domain, and use these features provide the software product line architecture .In other words, "FORM is a systematic method that focuses on capturing commonalities and differences of applications in a domain in terms of "features" and using the analysis results to develop domain architectures and components" [14]. Kobra is a component-oriented approach to develop a product line based on the UML features. This method integrated the two paradigms into a semantic, unified approach to software development and maintenance [15]. This method can be used to support the Model Driven Architecture [12]. QADA or Quality-driven Architecture Design and Analysis, is a product line architecture design method that provides traceability between the product quality and design time quality assessment [12]. In fact, QADA method includes architectural design and quality analysis [16]. In this study the issues and challenges of designing a software product line is presented in terms of administrative and organizational aspects and technical aspects also described four software product line frameworks. Then, these frameworks are compared with

each other based on issues and challenges raised such that, each framework is improved which of the issues and challenges.

2. Issues and Challenges in Software Product Line Development

Product line architectures challenges which are not present single product architectures [17], and this made them different from the single product architecture. Issues and challenges in developing a software product line in [18] are evaluated from two aspects.

2.1. Administrative and Organization Aspects

For the develop software product line some organization factors should be examined and the results analyzed to obtain organizational success. Organizational factors and their impact on software product line can be categorized as follows:

- **Roles and responsibilities within an organizational structure:** Roles and responsibilities must be defined and classified to conform to the product line and end–product development processes. Successful software product line engineering requires management and coordination of the reusable components and end–product development projects to obtain the business objectives of the organization. Also the organizational structure should be compatible with the new roles and responsibilities. Changes in organization structure may cause resistance to change and reduce the motivation for their employees in the organization. Developing a software product line has a lot of problems due to its inherent complexity. Reusable components should be designed and developed by the domain engineering; also, these reusable components should be designed and integrated in the end-product by the application engineers.
- **Intergroup communication capabilities:** Mechanisms of interaction should be defined for an intergroup communication establishment. Interaction mechanisms should be defined in the development life cycle. "Since the development life cycles of the product line and the end–product development are strongly related to each other, a well–established inter-group communication will have a bigger contribution on the success of product lines than it would have in single product development projects"[18].
- **Changing thinking and continuous learning of the concept of software product line:** Product Line concepts should be introduced to staff up the specific features of the product line to be considered at all stages of the development life cycle. Changes required for improvement of the organizational culture should be considered.
- **Training:** Personnel should be trained on all aspects of the product line, such as architecture, processes and methodologies. Regularly so that training programs should be done to increase organizational knowledge.
- **Adapting new technologies in the product line:** Development of current and future products planned in the near future should be aligned with the advancement of technology. Continuous improvements are needed to accommodate new technologies in the product line, to maintain competitive in related products.
- **Strategic plans of the organization:** Higher level of monitoring and tracking are needed to determine a new strategic direction for the introduction of new product families in the market. Product line development strategy should be added to organization's functional areas of business and long term organizational objective. Strategic planning should be clearly outline what is to be developed from the software product line in order to gain competitive advantages and capture market segments to achieve strategic targets.

- **Marketing strategies:** Marketing Strategies should be developed for favorite product with the most appropriate timing for the not to miss market opportunities. Generally, product lines serve to a specific domain. For obtaining a competitive advantage in the market, and satisfy the needs of this domain, big initial investment might be required. Takes a long time for developing a mature product line; which means that the revenues might not be realizable in near or even mid-terms in some cases. Hence, the time required to deliver the product to market is very important. So, software product line development, in the long term, can be a more economical choice. However, developing reusable core artifacts could not be amortized, if only a few products with little commonality will be produced.

2.2. Technical Aspects

In this section we will try to discuss the challenges in the development of the product line. A key difference between traditional reuse and ideas of the product line is variability, which means that the reusable assets of a product line should be designed explicitly to obtain differences. Variability has impact on all artifacts, from requirements to code.

- **Requirements:** In requirements, types of variability can exist in product line includes: a requirement may be mandatory or optional for all or some of the end-products, a requirement might depend on other requirements, a requirement may be incompatible with other requirements, and a requirement may have variations in details. Each item listed in the specific context of practical difficulties, and requirements analysis techniques used in traditional development cannot be implemented in all these variations. There is no systematic process for identifying and managing conflicts of requirements in the case that the requirements are written in natural language. Formal specification languages that were provided for this purpose are very complex and difficult for non-experts.
- **Design:** It is important to architecture design to be less dependency between modules to be too much trouble during the development and maintenance. This main factor is difficult to achieve, especially for cases of product line migration. Another design problem encountered in practice is the so-called design erosion. Means that the initial development that is not enough to consider all aspects of design and various external factors such as timing Pressure and intolerances of management to product line cost and time overhead, etc. Design erosion, integration and maintenance costs will increase and will be working again. Another important problem, more complex software systems must be developed by a multi-tier architecture that can often be implemented in future product line. Currently, not much data about multi layer product lines compositions.
- **Implementation:** For the programming stage of the development phase, if a lot of code complexity metrics seem to be also applicable to product lines at first glance, some of which may be misleading to monitoring the progress. Therefore, other metrics such as evaluation rate, dependencies between modules based on complexity, etc. should be defined and used. There are not many research studies about the use and importance of existing metrics in product lines or definition of new product line specific metrics.
- **Test:** Due to the variability of requirements, verification is generally more complex than a single product. For unit testing, test cases must be carefully selected and the logical way in order to minimize the time to testing and to avoid the error of error detection. Integration Testing due to the variability and dependencies between different products is more challenging. It should be designed carefully and often as an iterative integrated activity. Moreover, regression testing, which is often a confusing topic even for a single product development, becomes more complex in the product lines.
- **Maintenance:** High level of reuse in product line causes increasing challenges in the maintenance. As long as a product line evolves, reuse traceability is weak due to activities such as bug fixing,

functionality addition, performance optimization, etc. Another important problem is that maintenance activities are typically based code. Since the code-centric approach to avoid seeing the full picture of the inter- and intra-module dependencies, Special care should be given in order to spread the impacts of maintenance activities of all related products, appropriately. The ideal way to obtain such spread is cross–product line review for all change requests. So this is impractical and requires the participation of a large number of personnel from different teams; and hence, such studies need to spend time and cost.

3. Overview of Software Product Line Frameworks

Framework 1: DRAMA

DRAMA is a framework for domain requirements analysis and modeling architectures in software product lines. The goal is to provide a framework for modeling domain architecture based on the domain requirements within product lines. This study focused on the relationship between requirements and architectural structures. This framework includes processes, methods and a supporting tool [19]. The framework uses the four concepts, includes, goal-oriented domain requirements analysis, Analytical Hierarchy Process, Matrix technique, and architecture styles [19]. This framework is shown, in Fig. 2 similar to [19]. The DRAMA framework, in [19] is presented as follows:

1) Identifying components that make up the architecture, is obtained of the requirement analysis. Goal modeling used to elicit and to identify components according to four abstraction levels of requirements: business level, service level, interaction level, and internal level. At the end of this phase, the components are constructed.
2) Calculating the priority of components: the priority of components is calculated through AHP (Analytic Hierarchy Process) to understand which components have a major role.
3) Calculating the priority of quality attributes: the matrix technique is used to obtain the priority of each quality attribute. (the matrix technique is to identify the relationship between six quality attributes and components and to calculate how each component corresponds to the six quality attribute)
4) Modeling domain architectures: an architecture structure is obtained based on quality attributes. Domain architectures are built through components and quality attributes using architecture styles.

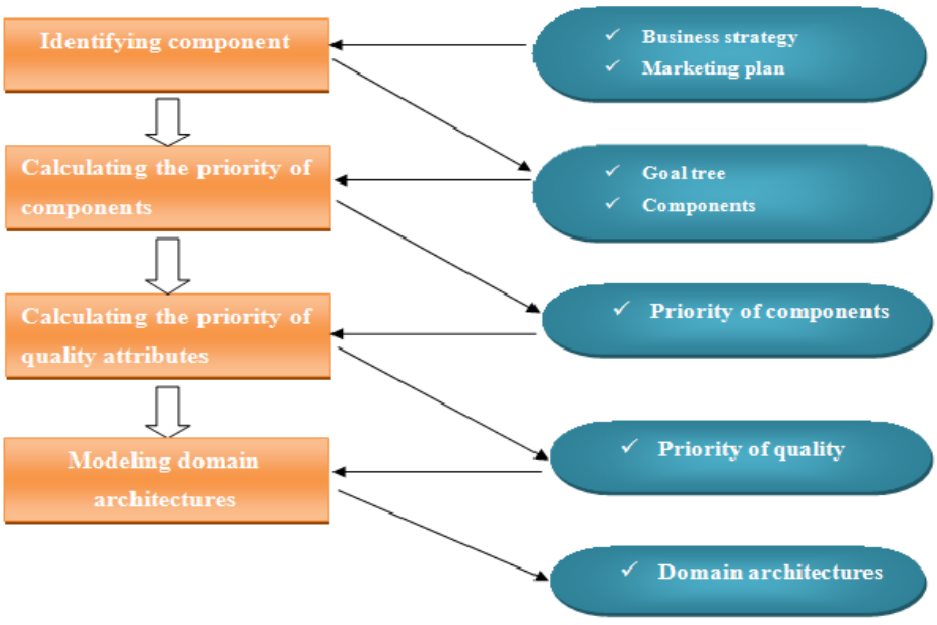

Fig. 2. Architecture of framework.

Framework 2

This framework is an architecture-based method for selecting composer components to make a software product line. This method can manage and control the complexities of the component selection problem in the creation of declared the product line [20]. With this method a product line will be constructed with accepted good components to cover up requirements based on the architecture [20]. That's why software product line development process reduces risks and costs of development. The method constructed in this framework is based on component oriented design. The framework built in architectural platform for the selection of a component with respect to different aspects related to reference architecture, product line requirement, domain requirement and priorities of the stakeholders [20]. Architecture of Method in [20] is as follows:

1) In Fig. 3, similar to [20], the idea of the method in selecting of the components is presented. The method surrounded by reference architecture because the reference architecture has a high impact on the selection of the components. Reference architecture defines requirements for components on the product line and limiting them to follow the rules. In the beginning step, a components list should be selected from component lists with respect to the architecture variant point. The components of the list can be selected from COTS components (COTS components are on the shelf that are produced by our organization or other entities.) or previously–produced components or be produced specially for the product line.

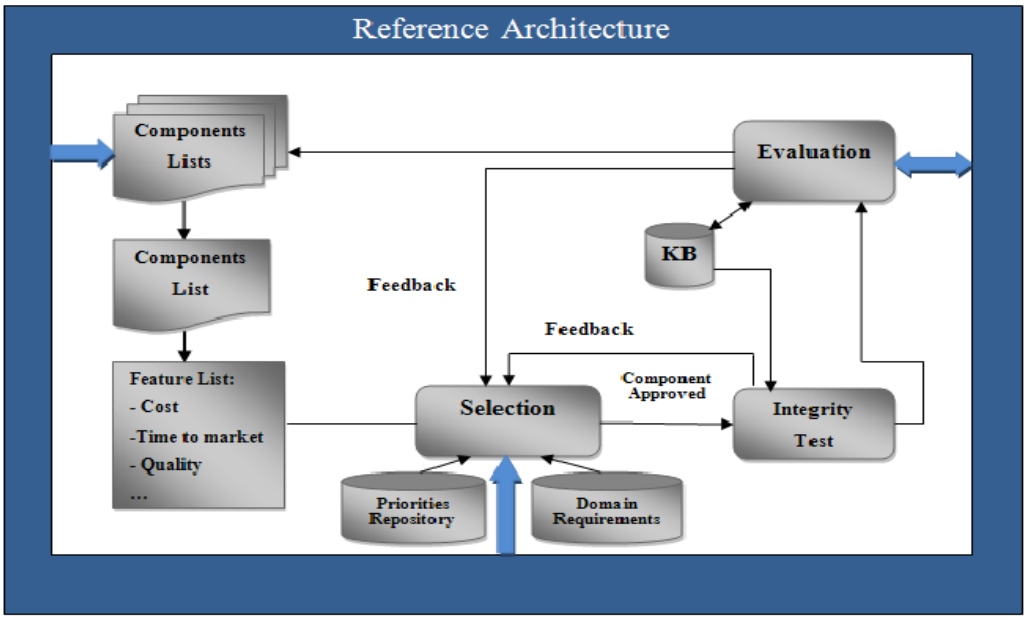

Fig. 3. Architecture of method.

2) In the next step, selected components will be evaluated through a feature list such as cost, time to market and quality attributes. This information is used for component selecting in the selection step. Also, the need is to respect the reference architecture in selection step. In this step, some of the information is extracted from domain requirements and some from priorities repository. Domain requirements concern about special domain requirements and priority repository concerns about the organization priorities. If the component is approved, method will go to integrity test step.
3) In the integrity test step, selected components are checked together; these components are stored in the KB. This means, a new component is checked against KB that they do not have a conflict in requirements. The feedback from this step goes to the selection step in cases which the selected component is removed from integrity test and there is a need for the newly selected component.

4) In the evaluation step, the architecture will be evaluated according to the selected components. In this step, it is possible to change the architecture to adapt to the requirements of the selected components and increase product quality and reduce cost. Also, the removed components are considered and architecture is evaluated. The feedback from this step goes to the selection step.

There is a two-way relationship between evaluation step and reference architecture; because, there may be errors in design of the architecture. After evaluation step, the method returns to the first step and selects a new component. Selection process continues until selection of the all components.

Framework 3

In this framework security mechanisms served to the domain requirements and application requirements of software product line. In fact the product line has been enriched. This framework in [21] is presented as follows:

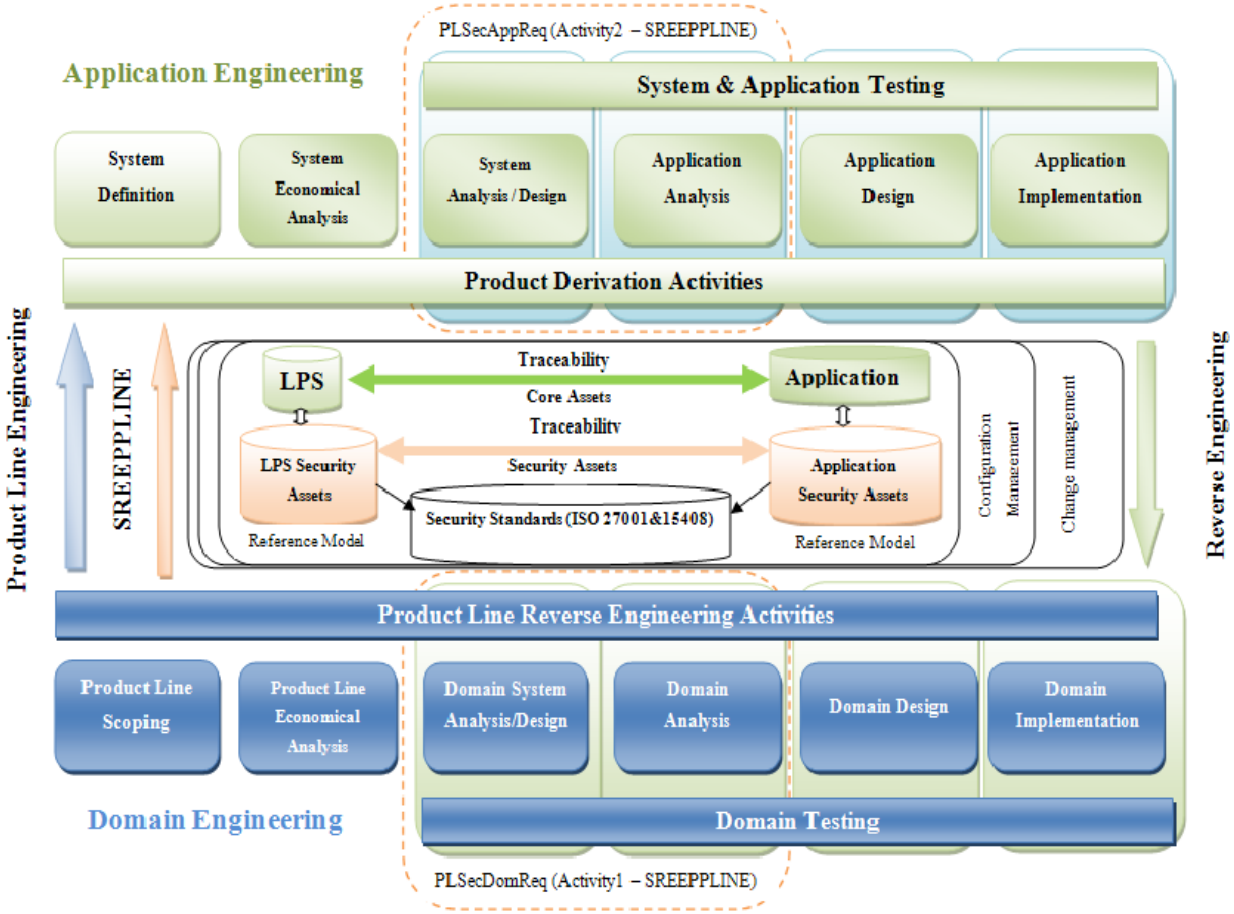

Fig. 4. Framework for software product line.

This framework includes two types of activities : application engineering (at the top of the figure) and domain engineering (at the bottom of the figure). This framework is shown, in Fig. 4 similar to [21]. The integration of the product line security domain requirements engineering (PLSecDomReq) and product line security application requirements engineering (PLSecAppReq) SREPPLine activates within activates of domain engineering and application engineering is shown in the figure with a dotted line. SREPPLine activities should be integrated in the domain analysis activities and application engineering processes in an organization where the software development idea is based on the software product line engineering. In the center of the figure is an arrow indicating the direction of SREPPLINE is the same path product line engineering that should be integrated into software product line engineering. It is not designed for integration into a reverse engineering process. Repositories, traceability, and implementation of the

security reference model as shown in the center of figure. The link between the software product line repositories and the repositories of its derived from applications are also shown. The repositories are:

1) Software product line repository: It keeps the common artifacts of the product line and the links with the artifacts of their derived applications.
2) Application repository : There is a repository for all application types that are derived from the product line. This repository registers the artifacts derived from the product line and the application's Specific artifacts
3) Software product line Security Assets repository : This manages the product line security artifacts such as security objectives, threats, security requirements, etc. and the relations with the security artifacts of their derived application ,the security standards and Related artifacts of the product line such as, accessibility requirements.
4) Application Security Asset repository: This repository manages security artifacts derived from the product line and the specific security artifacts of the application. It also manages the links with the related artifacts of the application and the security standards.
5) The security standards repository registers the security standards such as ISO/IEC 27001 controls or ISO / IEC 15408 components.

Framework 4

This framework is for software product line engineering comprises the concepts of the traditional product line engineering, namely the use of the platform and the ability to provide mass customization [22].

The idea of software product line engineering is composed of two separate processes. The idea of the framework is shown, in Fig. 5 similar to [22]. This framework in [22] is presented as follows:

1) Domain engineering: This process is responsible for creating reusable platforms, thereby defining the commonality and the variability of product line software. The platform includes all software artifacts such as requirements, design, realization, tests, etc.

Traceability links between these products facilitate consistent and systematic reuse. Domain engineering process, a combination of five key sub-processes, including product management, domain requirements engineering, domain design, domain realization, domain testing.

2) Application engineering: This process is responsible for mining product line applications from the platform created in domain engineering. This process is a combination of the sub-processes application requirements engineering, application design, application realization, application testing.

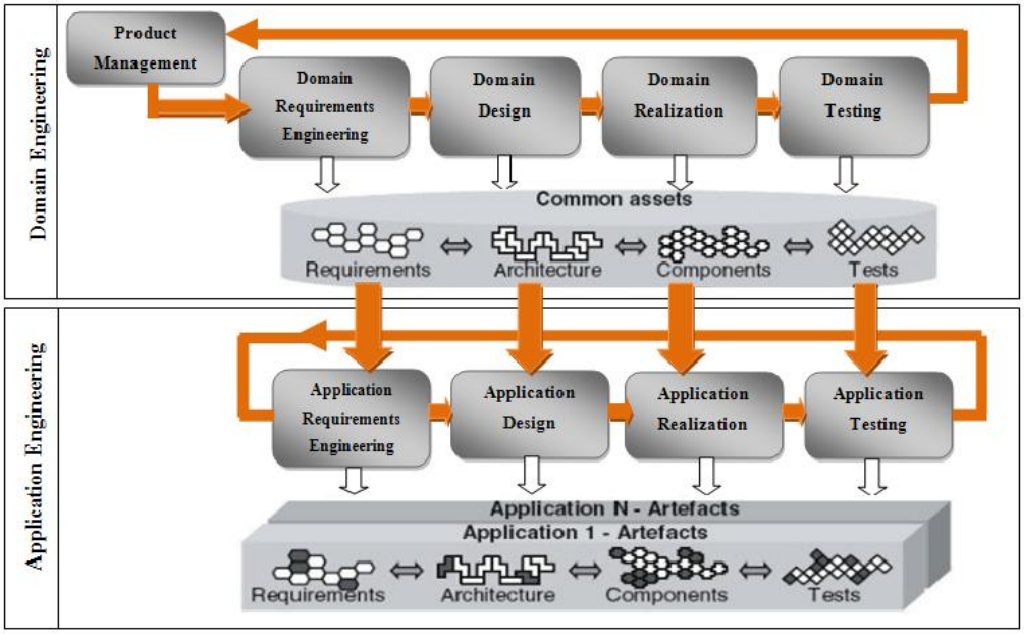

Fig. 5. The software product line engineering framework.

4. Discussion

This section has compared four frameworks for software product line and checks each framework improves which of the issues and challenges of software product line. Each of the frameworks under evaluation has a specific purpose and some of the challenges of product line covered. The overall goal of these frameworks is the produce architecture of software product lines.

DRAMA framework is built based on the FORM framework. One of the technical challenges of software product line is about requirements. With regard to questions such as: "How can we analyze domain requirements for modeling domain architecture? How can we design domain architecture corresponding to domain requirements? How can we assure that domain architecture can be reconfigured according to changing requirements?"[19] The answer to these questions is the use of traceability between domain requirements and domain architecture [19]. Much research in this area has been done and provided the many frameworks for the areas that could not support the challenge even FORM framework is a feature-oriented reuse method can not to support relationships between requirements and architecture. DRAMA is a framework that improves the requirements challenge because one of the main goals of this framework is the traceability between requirements and domain architecture. This means that changes in users' needs are reflected in the architecture. Thus, this framework can be useful to obtain better time-to-market. From this viewpoint can in the organizational and administrative aspects the improve issues of organization strategic plans and marketing strategies. Validation of the framework is performed with a home integrated system product line.

Table 1. Frameworks Influence on Issues and Challenge

Issues and Challenges	Frameworks			
	Framework 1 (DRAMA)	Framework 2	Framework 3	Framework 4
Strategic plans of the organization	✔	✔	✔	✔
Marketing strategies	✔	✔	✔	✔
Requirements	✔		✔	✔
Security requirements			✔	
Design		✔		✔
Implementation				
Test				
Maintenance		✔		

Framework 2 was developed based on the basic framework of COPA. This framework according to the characteristics of COPA, a component-oriented software product line development. This framework aim is control and manages the complexities of the component selection problem in the product line. According to the characteristics of component reuse and a list of features that are considered in the framework for selecting the components included reducing the cost and time-to-market, this framework can reduce the cost and time-to-market in software product line. So from this point of view can improve the issues of organization strategic plans and marketing strategies. Also the use of components reduces the dependency between them; hence this framework improves the challenges related to the design. On the other hand, if made a high level of reuse of components increases the challenge of Maintenance in the product line.

Validation of the framework is illustrated with electronics industry product line like TV.

Framework 3 is built on a base FAST framework. Its main goal is to provide the security quality requirements in software product line. If requirements be considered as one of the technical challenges of product line, providing security requirements specification for product line can be more challenging for the product line hence, this framework has improved the challenge. The framework was validated by a case study in the field of electronic billing.

Framework 4 is built on a base FAST framework. This framework provides a platform to facilitate mass customization to meet the needs of various stakeholders. For this purpose, the concept of variability is introduced in the platform. This platform is a powerful platform for creating customer applications in a short time. In this framework, based on input from Product Management, domain requirements engineering predicts future changes in requirements, such as regulations, standards, and changes in technology and market requirements for future applications. Hence, this framework can be improved challenge of the requirements. It can also improve the issues of organization strategic plans and marketing strategies. Also the result of the domain realization is consists of loosely coupled, configurable components. It can improve the design challenges in terms of reducing the dependencies between components. According to the discussion section of the paper, Table 1 presents the influence of frameworks on the issues and challenges for software production line.

5. Conclusion

By combining the application of these frameworks, the key results are as follow.

One of the main goals of the architecture of software product lines and the four frameworks is to reduce costs and time-to-market. Hence, these frameworks can improve the issues of organization strategic plans and marketing strategies. DRAMA framework, can establish traceability between requirements and architecture and framework 4 with a strong platform to meet the needs of various stakeholders can improve the challenges related to requirements. Framework 3 can enrich the product line and improve the challenges of security requirements in the product line. Framework 2 and framework 4, can improve the design challenge, to reduce the dependencies between software components. In framework 2, high-level reuse of components may give rise to challenges relating to Maintenance.

6. References

[1] Parnas, D. L. (1976). On the design and development of program families. *Software Engineering*, 1-9.

[2] Bass, L., Clements, P., & Kazman, R. (2003). *Software Architecture in Practice*. Addision Wesley.

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

[4] Anquetil, N., Kulesza, U., Mitschke, R., Moreira, A., Royer, J. C., Rummler, A., *et al.*, (2010). A model-driven traceability framework for software product lines. *Software & Systems Modeling*, *9(4)*, 427-451.

[5] Clements, P., & Northrop, L. (2002). *Software Product Lines*. Addison-Wesley.

[6] Gomaa, H. (2006). Designing software product lines with uml 2.0: From use cases to pattern-based software architectures. *Proceedings of Conference on 2006 10th International the Software Product Line* (pp. 218-218).

[7] Van, D. L. F. J., Schmid, K., & Rommes, E. (2007). *Software Product Lines in Action*. Springer-Verlag Berlin Heidelberg.

[8] Andrade, H. (2013). *Software Product Line Architectures: Reviewing the Literature and Identifying Bad Smells*. Master dissertation, Malardalen University, Vasteras.

[9] Van, G. J., Bosch, J., & Svahnberg, M. (2001). On the notion of variability in software product lines.

Proceedings *Working IEEE/IFIP Conference on. In Software Architecture* (pp. 45-54).

[10] Bosch, J., Florijn, G., Greefhorst, D., Kuusela, J., Obbink, J. H., & Pohl, K. (2002). Variability issues in software product lines. In F. van der Linden (Eds.), *Software Product-Family Engineering* (pp. 13-21).

[11] Deelstra, S., Sinnema, M., & Bosch, J. (2004). Experiences in software product families: Problems and issues during product derivation.

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

[13] Weiss, D., Lai, C., & Tau, R. (1999). *Software Product-Line Engineering: A Family-Based Software Development Process*. Addison-Wesley.

[14] Kang, K. C., Kim, S., Lee, J., Kim, K., Shin, E., & Huh, M. (1998). FORM: A feature-Oriented reuse method with domain-Specific reference architectures. *Annals of Software Engineering*, *5(1)*, 143-168.

[15] Atkinson, C., Bayer, J., & Muthig, D. (2000). Component-based product line development: The KobrA approach. *Software Product Lines*.

[16] Matinlassi, M., Niemelä, E., & Dobrica, L. (2002). *Quality-Driven Architecture Design and Quality Analysis Method*. VTT publication.

[17] Garlan, D. (2000). Software architecture: A roadmap. *Proceedings of the Conference on the Future of* Software *Engineering* (pp. 91-101).

[18] Dura, O., & Yilmaz, A. E. (2009). Software product line development: A review on practical issues and challenges. *Proceedings of the 24th International Symposium on Computer and Information Sciences* (pp. 736-742).

[19] Kim, J., Park, S., & Sugumaran, V. (2008). DRAMA: A framework for domain requirements analysis and modeling architectures in software product lines. *Journal of Systems and Software*, *81(1)*, 37-55.

[20] Tanhaei, M., Moaven, S., & Habibi, J. (2010). Toward an architecture-based method for selecting composer components to make software product line. *Proceedings of the 2010 Seventh International Conference on Information Technology: New Generations (ITNG)* (pp. 1233-1236).

[21] Mellado, D., Fernández, M. E., & Piattini, M. (2010). Security requirements engineering framework for software product lines. *Information and Software Technology*, *52(10)*, 1094-1117.

[22] Böckle, G., Pohl, K., & Van, D. L. F. (2005). A framework for software product line engineering. *Software Product Line Engineering*.

Alireza Olyai is a master student in the Department of Software Engineering, Shahid Beheshti University. He is a lecturer at the University of Applied Science and Technology. His research interests are service oriented architecture, enterprise architecture, ultra large scale systems, systems integration, and database systems.

Reza Rezaei obtained his Ph.D. degree from the University of Malaya. His research interests are interoperability, cloud computing, and service oriented architecture. He is an assistant professor at the Department of Computer Engineering, Faculty of Technical and Engineering Sciences, Islamic Azad University of Saveh Branch, Iran.