Experience on Requirements Engineering for Software Product Line

Waraporn Jirapanthong

Faculty of Information Technology, Dhurakij Pundit University, 110/1-4 Prachachuen Road, Laksi, Bangkok 10210, Thailand, waraporn.jir@dpu.ac.th ,+66(0)9547300 ext. 385

ABSTRACT – Software product line has been recognised as an important paradigm for software systems engineering. However, it has been quested whether software product line-based approach is more productive and flexible than a single software development process. This paper thus describes the experiences and challenges of requirements engineering and management for software product line, in comparison with individual software systems. This study was conducted to identify the requirements engineering practices that clearly contribute to software project success. It investigated team knowledge, allocated resources, and deployed requirements engineering processes. To compare two software processes, software product line and classic individual software process, this research conducted an experiment involving three software development projects that have similar requirements and some different requirements.

KEYWORDS – Software Product Line; Product Family; Requirements Engineering and Management; and Requirements Development

---

1. Introduction

Requirements management is concerned with understanding the goals of the organisation and its customers and the transformation of these goals into potential functions and constraints applicable to the development and evolution of products and services. It involves understanding the relationship between goals, functions and constraints in terms of the specification of products, including systems behaviour, and service definition.

The goals represent why a certain extent relates and what are in development terms. The specification provides the basis for analysing requirements, validating what stakeholders want, defining what needs to be delivered, and verifying the resultant developed product or service.

Requirements management aims to establish a common understanding between the customers and stakeholders and the project team that will be addressing the requirements at an early stage in the project life cycle and maintain control by establishing suitable baselines for both development and management use.

In addition to, many software development projects focus on customer satisfaction, quick adaptation to changes, and flexibility. Therefore, software product line development has become popular because it responds well to frequent changes in user requirements. Software product line shares a common set of features and are developed based on the reuse of core assets have been recognised as an important paradigm for software systems engineering. Recently, a large number of software systems are being developed and deployed in this way in order to reduce cost, effort, and time during system development. Various methodologies and approaches have been proposed to support the development of software systems based on software product line development.

However software product line development is criticized as having difficulties. Some difficulties are concerned with the (a) necessity of having a basic understanding of the variability consequences during the

different development phases of software products, (b) necessity of establishing relationships between product members and product line artefacts, and relationships between product members artefacts, (c) poor support for capturing, designing, and representing requirements at the level of product line and the level of specific product members, (d) poor support for handling complex relations among product members, and (e) poor support for maintaining information about the development process.

This paper thus describes the experiences and challenges of requirements engineering and management for software product line, in comparison with individual software systems.

2. Background of Requirements Engineering and Management for Software Product Line

Software product line systems share a common set of features and are developed based on the reuse of core assets. Many approaches [1],[2],[4],[7],[8] have been proposed to support software product line development. Those approaches mainly focus on domain engineering which is concerned the identification and analysis of commonality and variability principles among applications in a domain in order to engineer reusable and adaptable components and, therefore, support product line development. There are three steps for domain engineering:

(a) domain analysis is the process of identifying, collecting, organizing and representing the relevant information in a domain, based upon the study of existing systems and their developing histories, knowledge captured from domain experts, underlying theory, and emerging technology within a domain [7]. Software artefacts that are produced during the activity of domain analysis are called reference requirements, which define the products and their requirements in a family. The reference requirements contain commonality and variability of the product family. The activities occur during the domain analysis are scoping, defining of commonality and variability, and planning for product members and features.

(b) domain design is the process of developing a design model from the products of domain analysis and the knowledge gained from the study of software requirements or design reuse and generic architectures. Software artefacts that are produced during the activity of domain design are called software product line architecture, which forms the backbone of integrating software systems and consists of a set of decisions and interfaces which connect software components together. Software product line architecture differs from an architecture of individual systems that it must represent the common design for all product members and variable design for specific product members. The activities occur during the domain design are defining and evaluation of software product line architecture.

(c) domain implementation is the process of identifying reusable components based on the domain model and generic architecture [3]. Software artefacts that are produced during the activity of domain implementation are called reusable software components. The activity is focused on the creation of reusable software components e.g. source codes and linking libraries that are later assembled for product members At the end of the domain engineering process, an organization is ready for developing product members.

Additionally, *application engineering* is a systematic process for the creation of a product member from the core assets created during the domain engineering. Domain engineering assures that the activities of analysis, design and implementation of a product family are thoroughly performed for all product members, while application engineering assures the reuse of the core assets of the product family for the creation of product members. There are activities such as: (i) *requirements engineering,* which is a process that consists of requirements elicitation, analysis, specification, verification, and management; (ii) *design analysis,* which is a process that is concerned with how the system functionality is to be provided by the different components of the system*;* and (iii) *integration and testing,* which is a process of taking reusable components then putting them together to build a complete system, and of testing if the system is working appropriately.

However, although the support for identifying and analysing common and variable aspects among applications and the engineering of reusable and adaptable components are important for software product line development, they are not easy tasks. This is mainly due to the large number and heterogeneity of documents generated during the development of product line systems.

For the development of individual software system, the process has five major activities: (a) requirement definition, (b) software and system design, (c) implementation, (d) integration and testing, and (e) operation and maintenance. The documents are created for each individual software system.

3. Research Method

3.1. Software Development Team and Projects

This study was conducted to identify the requirements engineering practices that clearly contribute to software project success. It investigated team knowledge, allocated resources, and deployed requirements engineering processes. To compare two software processes, software product line and classic individual

software process, we established two software development teams with the equivalent skills but worked on the same projects and used different processes.

The software development teams that have experience in software development were participated in the study. Each software development team was established and included a system analyst, a project manager, and four software developers. This research conducted an experiment involving three software development projects that have similar requirements and some different requirements. Three software projects were designed in order to narrow set of requirements and those are based on business applications. On average, the projects finished in 2.5 months with an effort of 6 person-months. The development team was required to achieve the software development projects: (i) one team was to follow the software product line process to complete those three software projects as a line, and (ii) another one was to follow any software process to complete the software projects.

3.2. Empirical Project Development based on Software Product Line Process

The project started with developers training in software product line processes and techniques. These developers were then tested for their understanding of software product line practices by using questionnaires. Those who passed the test were assumed to be ready to implement projects using software product line. At the beginning of a project the developers need to take several days to envision the high-level requirements and to understand the scope of the release. The goal of this activity is to find what the project is all about, not to document in detail. The developers then started developing a set of three projects by following the software product line practices. They studied and analyzed all projects together and produced the software artefacts: (i) reference requirements; (ii) software product line architecture; and (iii) software components.

The artefacts were checked before submitting to the domain repository to be ready for application engineering process. Next, three software products were created based on the domain artefacts (i.e. reference requirements, software product line architecture, and software components). Before the software was accepted by customers, we ran test cases on the software. When the software passed all test cases, the projects are completed. The whole software product line process is shown in Figure 1. We then calculated and analyzed the qualitative and quantitative aspects of domain engineering process and application engineering process for each project. Then we checked the developers conform to software product line practices.

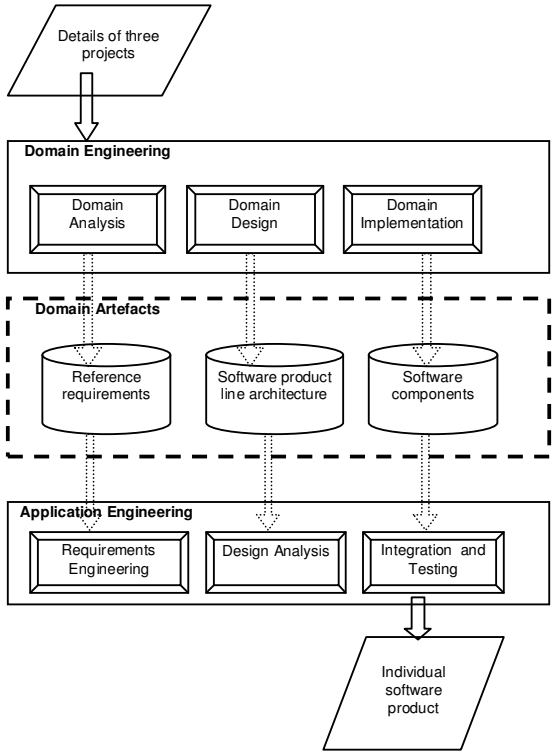

Figure 1. Software Produce Line Process

3.3. Empirical Project Development based on Individual Software Process

For each project, developers divided their work based on their roles. Firstly, the developers summarized all requirements from customers and produced a user requirement specification. Next, they designed the system architecture, components and data models. They applied use case descriptions and diagrams to explain the requirements of each individual software product. In addition, they also created class diagrams, sequence diagrams and activity diagrams of the entire project in this stage. They implemented the software by following the documents and used unit tests regularly. When completing all the components, the developers integrated all the pieces together and began an integration test. Finally, the developers delivered the customers the complete software when all of these stages finished. The artefacts that are checked and submitted to the repository are: (i) use case descriptions; (ii) use case diagrams; (iii) class diagrams; (iv) sequence diagrams; (v) activity diagrams; (vi) source code; (vii) testing documents; and (viii) coding standard and technical documents. All the end of this step, we calculated and analyzed the qualitative and quantitative aspects in each project.

4. Requirements Development for Software Products

4.1. Artefact Type

At the beginning of the experiment, the developers are given the description of artifact types which should be applied for requirement engineering process. However, in practical, there are several different types of requirements. Each modeling artifact has its strengths and weakness. We then propose a set of artifact types which are applied to capture and specify the software requirements. Therefore several requirements modeling artefacts are applied. Table 1 summarises common artefacts for modeling requirements in projects. Each artifact type is applied in order to make the requirements precise and consistent. Some of these methods have separated the data, functional, and behavioral aspects of requirements and specified software by creating one or more distinct models. Prototypes, for instance, attempt to create an operational model that stakeholders or users can directly experience.

Followed by many other models, the development team found missing and inconsistency of some requirements. Thus this set of artifact types is integrated and focused on establishing of a closer link between models and business goals. As shown in Table 1, each artifact type is applied to specify either behavioral or non-behavioral requirements, and captured by different simple tools.

4.2. Requirement Development for Software Product Line and Individual Software Systems

According to the software product line approach, the projects have been developed based on study, analysis, and discussions of business domain. The team of developers analysed and designed a family of software systems with three members. Each member has shared and specialized functionalities with the family. The product members are aimed to satisfy different targets of customers. As shown in Table 1, several types of requirements artefacts are created and specified the requirements. Moreover, those artefacts are further developed according to the software product line approach and the models [5] are created to represent the software product line. Particularly, the reference requirements is produced and documented in term of a feature model as software product line architecture is produced and documented in terms of subsystem, feature, and process models [5]. The feature model is created and composed of common features representing mandatory features, alternative and optional, representing different features between product members. The subsystem models is created and provides facilities for performing basic tasks in the systems. But there exist various instances of the process and module models, as well as there exist many instances of use cases, class, statechart, and sequence diagrams. The process models are created and each is refined for a subsystem in the subsystem model. The module models are created and each is refined for a process in the process models. Moreover, the artefacts of each product member are created. For example, a use case is used to elaborate the satisfaction of the functionalities for each product member.

For individual software development, requirements for each software project is captured according to the artifact type in Table 1 and further developed to be a number of artefacts for each individual software product during individual software development process. Finally, the software artefacts of each individual software product are usecase diagram, use case descriptions, class diagrams, statechart diagrams, sequence diagrams, and source code.

During the activities of requirements elicitation in order to transform the requirements to be other software artefacts for each product, there are several techniques applied. We summarised in Table 2.

4.3. Change Management for Software Product Line and Individual Software Systems

As mentioned earlier, this study was conducted to identify the requirements engineering practices that clearly contribute to software project success, comparatively between software product line and individual software systems. The experiment involved three software development projects, PM1, PM2, and PM3 that have similar requirements and some different requirements. Furthermore, the study was conducted to handle a situation of change management on requirements. There was a set of change on the existing software products.

According to the software product line-based systems, new requirements management can be facilitated by the identification and analysis of commonality and variability principles among software product line and product members. A number of relations between artefacts are detected in order to determine the association between the new requirements and existing software artefacts in product member and software product line. Different types of traceability relations are created to identify the role of those relations [6].

Table 1. Common artefacts for modeling requirements

Artifact	Type	Simple tool	Description
Acceptance test	Either	Paper	Describes an observable feature of a system which is of interest to one or more project stakeholders.
Business rule definition	Behavioral	Index card	A business rule is an operating principle or policy that software must satisfy
Constraint definition	Either	Index card	A constraint is a restriction on the degree of freedom that a developer team have in providing a solution. Constraints are effectively global requirements for a project.
Data flow diagram (DFD)	Behavioral	Paper	A data-flow diagram (DFD) shows the movement of data within a system between processes, entities, and data stores. When modeling requirements a DFD can be used to model the context of the system, indicating the major external entities that the system interacts with.
Essential UI prototype	Either	Draft paper	An essential user interface (UI) prototype is a low-fidelity model, or prototype, of the UI for the system. It represents the general ideas behind the UI but not the exact details.
Essential use case	Behavioral	Paper	A use case is a sequence of actions that provides a measurable value to an actor. An essential use case is a simplified, abstract, generalized use case that captures the intentions of a user in a technology and implementation independent manner.
Feature	Either	Index card	A feature is a small useful result in the perspective view of users. A feature is a tiny characteristic of the system. It is understandable, and do-able.
Technical requirements	Non-behavioral	Index card	A technical requirement pertains to a non-functional aspect of the system, such as a performance related issue, a reliable issue, or technical environment issue.
Usage scenario	Behavioral	Index card	A usage scenario describes a single path of logic through one or more use cases or user stories. A use case scenario could represent the basic course of action.
Use case diagram	Behavioral	Draft paper	The use case diagram depicts a collection of use cases, actors, their associations , and optionally a system boundary box. When modeling requirements a use case diagram can be used to model the context of the system, indicating the major external entities that the system interacts with.
User story	Either	Index card	A user story is a reminder to have a conversation with the project stakeholders. User stories capture high-level requirements, including behavioral requirements, business rules, constraints, and technical requirements.

Table 2. Techniques for eliciting requirements

Technique	Description	Strength(s)	Weakness(es)
Active stakeholder participation	Extends on-site user to have stakeholders (users) actively involved with the modeling of their requirements.	- Highly collaborative technique - Domain expert can define the requirements - Information is provided to the team in a timely manner - Decisions are made in a timely manner	- Many stakeholders need to learn modeling skills - Stakeholders are not available full time
Face-to-face Interview	Meets key stakeholders to discuss their requirements.	- Collaborative technique - Developers can elicit a lot of information quickly from a single person - Stakeholders can provide private information that they would not publicly tell	- Interviews must be schedules in advance - Interviewing skills are difficult to learn
Reading	A wealth of written information available from which developers can discern potential requirements or just to understand stakeholders better.	- Opportunity to learn the fundamentals of the domain before interacting with stakeholders	- Restricted interaction technique - Practical usually differs from what is written down - There are limits how much developers can read, and comprehend the information

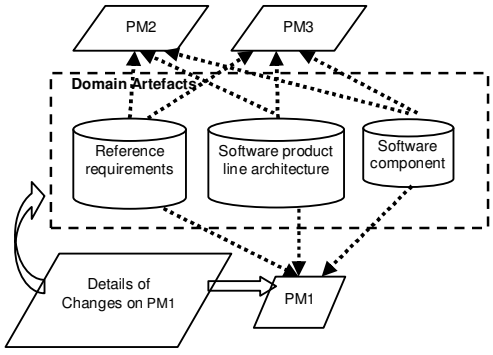

Figure 2. Change Management for Software Product Line System

For the software product line-based systems, it is supposed the situation in which the organization has established a software product line for their software systems with software product members. Those are created from the development phase. And the new requirements are done to a product member. Therefore, it is necessary to evaluate how these new requirements will affect the other artefacts of the product member and if these new requirements also affect other product members in the software product line that may be related to the new requirements. The artefacts are inspected and determined if they are related to the new requirements as shown in Figure 2. The change on software product member, PM1, was related to some of reference requirements, of software product line architecture, and of software components. During the activities of change management, the development team must have evaluated whether and how those changes would affect other software product members, PM2 and PM3. The team also must make a compromise between existing software products and new requirements in order to maintain the consistency of software requirements.

According to the individual software systems, it is necessary to evaluate how these new requirements will affect any artefacts of each software product. Developers divided their work based on their roles. They reproduced new user requirement specification and redesigned the system architecture, components and data models. They applied use case descriptions and use case diagrams to explain the new requirements of the software product. They updated class diagrams, sequence diagrams and activity diagrams of the entire project in this stage. They re-implemented the software by following the documents and used unit tests regularly. When completing all the components, the developers integrated all the pieces together again and began an integration test. Those activities were done to all individual software products. Finally, the developer delivered the customers the complete software when all of these stages finished.

5. Discussion and Conclusion

Requirements engineering and management is a central task of software product line development. It must be capable of deal with factors like upfront development of a domain model, the constant flow of requirements, a heterogeneous stakeholder community, a complex development organization, long-term release planning, demanding software architecture, and challenging testing processes. For successful software product line development, a collection of essential requirement development practices must be in place, which needs to support the meta project management capabilities. Many requirements engineering and management practices must be tailored appropriately to the specific demands of software product lines. The software engineering literature has pointed out the software product line development is more complex and demanding than individual software product development. This complexity has also particularly impact on requirements engineering and management. Of course, general challenges of requirements engineering and management also reoccur in software product line.

This work experienced the requirements engineering and management that arise in the context of industrial software product line development. To measure the project effort between using software product line and a single software process, we evaluated the number of entities that were created when the software products were developed and that were changed when there was new requirements. The entities are such as software design, software design specification, software code, and software development team. Attributes are such as defects discovered in design review, number of pages, number of line of code, number of operations, and team size, average team experience. In particular the size measure of software products involves two ratios: (a) line of code (LOC) and (b) function point (FP). As shown in Tables 3 and 4, the number of LOC created for each product members (PM1, PM2, and PM3) are 3689, 1251, and 2280. Moreover, there are 7280 LOCs additionally created for software product line artefacts. The numbers of LOC created for each individual software product are 6830, 5420, and 8845. However, the numbers of function point between same software products using different software processes are the same.

Additionally, the quality measure of software product involves maintainability measurement such as coding effort, design effort, percentage of modules changes, classes changes, classes added. In addition, we measured the maintainability metrics in external view such as mean time to implement the changes. Based on the same set of changes, it is found that the mean time to implement the changes on product member, PM1, which is created by using software product line process is 15.5 days. Comparatively, it is found that the mean time to implement the changes on software product, PM1, which is created by using a single software development process is 24 days. It is thus believed that well- and proper-implementation of software product line will be effective to maintainability.

Moreover, we also compare the development team's satisfaction. We conducted the survey and interview. The developers are observed for the satisfaction regarding the process of software product line. It is found that the developers are satisfied the process that emphasis the software more than the documentation. However, the process would be difficult to inexperience developers and some experience developers tend to resist some software product line practices. According to the survey, it is found that 33% of developers tend to resist software product line practices with the above reasons, whereas 70% of developers are positive to using software product line practices. Particularly, 82% of developers are satisfied when performed the maintenance phase with software product line. Some of

software product line artefacts are used during the maintenance phase. And it is satisfied by the developers. However, application engineering process depends on developer' skill. Moreover, some developers are unsatisfied to frequently update the documentation.

Table 3. The details of LOC and FP created for software products using software product line

Product Member	LOC created	FP created
PM1	3689	10
PM2	1251	5
PM3	2280	4
Software Product Line Artefacts	7280	n/a

Table 4. The details of LOC and FP created for software products using a single software process

Individual Software Product	LOC created	FP created
PM1	6830	10
PM2	5420	5
PM3	8845	4

Additionally, the developer teams found that types of requirements can be separated into two categories: behavioural and non-behavioural. A behavioural requirements describes how a user will interact with a system concerning user interface issues, how a user will use a system or how a system fufills a business function or business rules. These are often referred to as functional requirements. A non- behavioural requirements describes a technical feature of a system, features typically pertaining to availability, security, performance, interoperability, dependability, and reliability. Non-behavioural requirements are often referred to as "non-functional" requirements. It is very important to understand that the distinction between behavioural and non-behavioural requirements is fuzzy. A performance requirement which describes the expected speed of data access is clearly technical in nature but will also be reflected in the response time of the user interface which affects usability and potential usage. Access control issues, such as who is allowed to access particular information, partly is a behavioural requirement although they are generally considered to be a security issue which falls into the non-behavioral category. The critical thing is to identify and understand a given requirement. We found that it becomes an issue if the requirements are managed and mis-categorised.

Moreover, the results show that the effort metric of software product line-based projects is less than individual software projects. Software product line-based projects enhance the productivity by using existing software artefacts. The methodology supports software reuse at the largest level of granularity. However, developers spent time and effort to establish domain artefacts. Also, some defects are discovered during the integration process for a product member. It took some effort to fix them. On the other hand, in the single software team, customers are involved at the inception of project determined requirements and contractual agreement. Developers wrote all documents before coding. Then customers changed some requirements, maybe after they acquired finally product, developers needed to significantly redesign and edit their documents. This took a lot of effort to achieve the task.

However, software product line is unsuitable for all projects. It serves the reuse practice in an organization having a large number of products, which have similar requirements and some differences. Developers must consider the characteristics of the project to ensure software product line is appropriate. In the other hand, waterfall process is suitable to serve a software project which is small and has solid requirements. Also, the developers are responsible for estimating the effort required to implement the requirements which they will work on. Although the developers may not have the requisite estimating skills, it does not take long for them to get better at estimating when they know and get familiar with the software process methods.

References

[1] Atkinson, C., J. Bayer, and D. Muthig. 2000. *Component-based product line development: The KobrA approach*. Pages 289-310. the 1st Software Product Line Conference, SPLC. Kluwer, Denver, Colorado, USA.

[2] Bayer, J., O. Flege, P. Knauber, R. Laqua, D. Muthig, K. Schmid, T. Widen, and J.-M. DeBaud. 1999. *PuLSE: A methodology to develop software product lines*. Pages 122-131. the Fifth ACM SIGSOFT Symposium on Software Reusability (SSR'99), Los Angeles, CA, USA.

[3] Clements, P., and L. Northrop. 2004. *A Framework for Software Product Lines Practice*. http://www.sei.cmu.edu/productlines/framework.html

[4] Griss, M. L., J. Favaro, and M. d. Alessandro. 1998. *Integrating feature modeling with the RSEB*. Pages 76-85 in P. Devanbu and J. Poulin, eds. the 5th International Conference on Software Reuse. IEEE Computer Society Press.

[5] Jirapanthong, W. 2008. *An Approach to Software Artefact Specification for Supporting Product Line Systems*. the 2008 International Conference on

Software Engineering Research and Practice (SERP'08), Las Vegas, Nevada, USA, 2008.
[6] Jirapanthong, W., and A. Zisman. 2009. *XTraQue: traceability for product line systems*. Software and System Modeling 8(1): 117-144 (2009).
[7] Kang, K., S. Cohen, J. Hess, W. Novak, and A. Peterson. 1990. *Feature-Oriented Domain Analysis (FODA) Feasibility Study*. Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA.
[8] Weiss, D. 1995. *Software Synthesis: The FAST Process*. the International Conference on Computing in High Energy Physics (CHEP), Rio de Janeiro, Brazil.