Research Report

On

Comparative Study between Software Product Line and Waterfall Process

By
Waraporn Jirapanthong, Ph.D.
Dhurakij Pundit University

This research has received a research grant from
Dhurakij Pundit University
2011

รายงานผลการวิจัย

เรื่อง

การศึกษาเชิงเปรียบเทียบระหว่างกระบวนการพัฒนาซอฟต์แวร์แบบโปรดักต์ไลน์และแบบน้ำตก

โดย

ผู้ช่วยศาสตราจารย์ ดร. วราพร จิระพันธุ์ทอง

รายงานการวิจัยนี้ได้รับทุนอุดหนุนจาก
มหาวิทยาลัยธุรกิจบัณฑิตย์
พ.ศ. 2554

Research Title	:	Comparative Study between Software Product Line and Waterfall Process
Researcher(s)	:	Waraporn Jirapanthong
Institute/Publisher	:	Dhurakij Pundit University
Year of Publication	:	2011
Total Pages	:	63

บทคัดย่อ

ซอฟต์แวร์โปรดักต์ไลน์ได้รับการยอมรับว่าเป็นกระบวนทัศน์สำคัญทางด้านวิศวกรรมระบบซอฟต์แวร์ ในช่วงหลายปีที่ผ่านมามีการนำเสนอระเบียบแบบแผนและวิธีการจำนวนมากสำหรับสนับสนุนการพัฒนาระบบซอฟต์แวร์ที่มีพื้นฐานการพัฒนาแบบโปรดักต์ไลน์ อย่างไรก็ตามการพัฒนาแบบโปรดักต์ไลน์ยังคงมีความยุ่งยากในเชิงปฏิบัติ จึงกลายเป็นคำถามว่าวิธีการพัฒนาระบบซอฟต์แวร์แบบโปรดักต์ไลน์นั้นก่อให้เกิดผลประโยชน์และยืดหยุ่นมากกว่าการพัฒนาที่ใช้แม่แบบการพัฒนาระบบซอฟต์แวร์แบบดั้งเดิมอย่างแม่แบบน้ำตกหรือไม่ งานวิจัยชิ้นนี้พิจารณาแง่มุมเชิงปริมาณและคุณภาพของการพัฒนาซอฟต์แวร์ที่ได้โดยเปรียบเทียบระหว่างการพัฒนาโดยประยุกต์ใช้โปรดักต์ไลน์และน้ำตก งานวิจัยนี้นำเสนอการทำโครงการเชิงศึกษาผ่านกระบวนการแบบโปรดักต์ไลน์และน้ำตก มีการสำรวจและสัมภาษณ์เพื่อวัดความพึงพอใจของผู้มีส่วนเกี่ยวข้องในการพัฒนาระบบซอฟต์แวร์ มีการวัดเวลาที่ใช้ไปและวัดข้อผิดพลาดที่เกิดขึ้นระหว่างขั้นตอนการพัฒนาและบำรุงรักษาระบบซอฟต์แวร์ นอกจากนี้งานวิจัยชิ้นนี้ได้อธิบายประสบการณ์และปัญหาที่เกิดขึ้นของวิศวกรรมความต้องการและการจัดการความต้องการระหว่างการดำเนินโครงการพัฒนาซอฟต์แวร์แบบโปรดักต์ไลน์และแบบเดี่ยว

Abstract

Software product line has been recognised as an important paradigm for software systems engineering. In the last years, a large number of methodologies and approaches have been proposed to support the development of software systems based on product line development. However, its context leads difficulties to software product line engineering in

practical. It has been quested whether software product line-based approach is more productive and flexible than traditional software development model i.e. waterfall model. This research thus examines the qualitative and quantitative aspects of software development which applies software product line and waterfall. The research presents the study on empirical projects based on software product line and waterfall processes. In particular, we conduct the survey and interview to capture the satisfaction of stakeholders and measure the effort spent and errors occurred during software development and maintenance phases. Moreover, the research describes the experiences and challenges of requirements engineering and management that arise in the context of industrial software product line development. It has been derived from the study on empirical projects based on software product line and single software development.

Acknowledgement

I am grateful to Dhurakijpundij University for the financial support for this research.

Declaration

Some of the material in this report has been previously published in the paper:

- Waraporn Jirapanthong, "Comparative Study Between Software Product Line and Waterfall Process," Journal of Information Science and Technology, Vol.1, No.1 , pp.1-8.

I grant powers of discretion to Dhurakijpandit University to allow this research to be copied in whole or in part without further reference to me. This permission covers only single copies made for study purposes, subject to normal conditions of acknowledgement.

Contents

List of Figures

List of Tables

Chapter I Introduction

1.1 Research Motivation

Nowadays, many software development projects focus on customer satisfaction, quick adaptaion 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.

Although software product line development is criticized as having difficulties, it has been more popular. 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 research, thus, examines the qualitative and quantitative aspects of software development using software product line, in comparison with those using a traditional software model, waterfall model. In particular, the study used both qualitative aspect that were collected from surveys and interviews of development and maintenance team and quantitative aspect that were measured from effort spent during the development and maintenance phases.

Additionally, the research is concerned with requirements management regarding both software development approaches. 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.

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

1.2 Problems Statement

The problems could be classified into two categories as follows:

1. Software product line development becomes more being concerned regarding its difficulties.

2. Software developers lack of tacit knowledge of applying software product line approach in practical.

1.3 Research Objectives

The objectives of this research project are as follows:

1. To investigate the strengths and weakness of two software development approaches i.e. software product line and waterfall model;
2. To experience requirements engineering process in both software product line and waterfall model; and
3. To improve the performance of software system development.

1.4 Scope of Work

1. In this research, a team of software developers is set up. It includes a software engineer, system analyst, and programmers.
2. Software engineer and system analyst have well-experience in object-oriented analysis and design. Particularly, they design a system based on object-oriented techniques e.g. use case diagram, UML diagrams.
3. Programmers have well-experience in object-oriented programming, e.g. in java language.
4. In this research, we establish a case study of software development that encompasses three software projects.
5. Those software projects have some similar and different requirements.
6. The team of software developers develops the software projects by applying with both software development approaches i.e. software product line and waterfall model.
7. This research is based on object-oriented methodologies.

The remainder of this report is organized in five chapters as described below:

Chapter 2 presents the review of software product line, particularly activities of software product line development, framework of software product line artefacts, mapping different perspectives between feature model and UML diagrams, and the review of waterfall approach.

Chapter 3 describes the research method applied in the research.

Chapter 4 contains a description of the experiments and analyses the experiences on the case study.

Chapter 5 discusses the conclusions and directions for future work.

Chapter II Literature Review

This chapter presents the review of software product line, particularly activities of software product line development, framework of software product line artefacts, mapping different perspectives between feature model and UML diagrams, and the review of waterfall approach.

2.1. Software Product Line

Software product line is originally introduced to serve the reuse practice in an organization having a large number of products, which drives issues such as highly expensive, complex, and tedious tasks. The idea of product line was motivated by the need to systematize a number of products more effectively and the fact that these products have a certain set of common and special functionalities. For example, a mobile-phone company has created a mobile-phone family that contains a set of mobile-phones. Some lower end mobile-phones have similar basic functionalities but different hardware capacities to offer competitive price. Region-based mobile-phones are designed for different transmission and signaling standards, depending on regional diversity; thereby, the company provides different functionalities of mobile phones to support different regions.

2.1.1. Activities of Software Product Line Development

The main activities of software product line development i.e.

(a) domain engineering, and

(b) application engineering.

Domain engineering is a systematic process for the creation of the core assets [Northrop, L. M. 2002]. There are three steps for domain engineering:

(i) 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 [Kang, K., at el. 1990];

(ii) 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 [Garlan, D. and M. Shaw. 1993.]; and

(iii) domain implementation is the process of identifying reusable components based on the domain model and generic architecture [Clements, P., and L. Northrop. 2004].

Application engineering is a systematic process for the creation of a product member from the software artefacts created during the domain engineering [Northrop, L. M. 2002]. The application engineering process is composed of three steps:

(i) requirements engineering focuses on identifying, collecting, organizing and representing requirements of a product member. The major difference between requirements engineering of an individual product and a product member is that stakeholders do not only focus on the specific product but also on the scope of product family;

(ii) design analysis is to analyse and design the architecture for a product member. Design analysis in application engineering must be consistent with the concept of design analysis in domain engineering; and

(iii) integration and testing 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.

The activities in domain engineering involve the creation of core assets which are expected to be used for all product members. As the activities in application engineering involve the reuse of the software artefacts to create a product member. As shown in Table 2.1, different types of software artefacts are created during the activities of software product line development. For example, reference requirements are created during domain analysis and represented for the requirements of a product line. Software product line architecture is created during domain design and represented for the architecture of the software product line while the architecture of a specific product is generated during design analysis of application engineering. As shown in Table 2.1, software artefacts are generated during the development of software product line systems.

Many methodologies and approaches introduce methodical support to the activities of software product line engineering. However, software product line developers suffer from some of the following shortcomings:

(i) uncontrolled growth of variety;

(ii) difficulty of defining commonality and variability;

(iii) difficulty of documenting management;

(iv) confliction and dependency between artefacts in product line systems;

(v) difficulty to specialise variability.

Table 2.1: Software artefacts created during software product line engineering

Software artefact	**Activity**	**Description**
Reference requirements	Domain analysis	Defining the products and their requirements of a family.
Software product line architecture	Domain design	Representing the architecture of software product line.
Reusable software components	Domain implementation	Being integrated with other reusable software components for a particular product member.
Specific-product requirements	Requirements Engineering	Specifying the requirements of a particular product member based on the reference requirements
Specific-product architecture	Design analysis	Representing the architecture of a particular product member based on the software product line architecture
Specific-product configuration and particular product member	Integration and testing	Integrated and configured reusable components that are conducted to be a particular product member

2.2. FRAMEWORK OF SOFTWARE PRODUCT LINE ARTEFACTS

In this section, the framework of software product line artefacts is presented.

2.2.1. Use Case Description

Many approaches proposed to apply use case description in the activities of software product line development. Moreover, we found some approaches that extend use case description for software product line engineering. In [Fantechi, A., at el. 2004], the authors proposed to express the requirements of product members of a product family by extending the use case definition given by Cockburn. The variability is expressed in use cases by using special tags. The tags indicate the requirements of a product family that need to be specialised for a product member. They proposed three types of tag:

(i) alternative tag, which specifies variable requirements with a predefined set of requirement variants;

(ii) parametric tag, which requires specifying of parameters to fill in a requirement for a product member, and

(iii) optional tag, which represents an optional requirement which can be instantiated. In [John, I., and D. Muthig. 2002], the authors extended use case specification by adding constructs for representing variant points for variable requirements and applied the decision model to express the relationships and dependencies between the variable requirements.

2.2.2. UML Modeling

Many existing approaches and methods apply UML modelling in software product line engineering. Some approaches such as [Clauss, M. 2001][Gomaa, H., and M. E. Shin. 2004] are proposed to adapt UML diagrams for modeling software product line. Gomaa [Gomaa, H., and M. E. Shin. 2004] proposed Product Line UML-based Software engineering (PLUS) by using UML modeling for the development of software product line. The author applied UML diagrams to represent the commonality and variability of software product line. In [Clauss, M. 2001], they use a UML class diagram to represent software product line architecture. They define three types of stereotypes for representing variability in a product family:

(i) variationPoint, which is used for a generalized class;

(ii) variant, which is used for a specialized class; and

(iii) optional, which is used for a class.

They applied two types of relationships to assist representation of variability:

(i) generalization/specialization, which associates between classes typed of variationPoint and variant; and

(ii) association with cardinality 0...1, which associates between any class and a class typed of optional.

Some work proposed the combination of patterns and discriminants to support representing of commonality and variability in software product line architecture. A pattern is represented by class and object diagrams and a discriminant is a feature representing a requirement that differentiates a system from another. They defined three types of discriminant:

(i) single discriminant, which represents a set of mutually exclusive features;

(ii) multiple discriminants, which represent a set of optional features which are not mutually exclusive; and

(iii) option discriminant, which is a single optional feature that may or may not be used.

The single discriminant represents an inheritance hierarchy that consists of a generic class called base class and a set of subclasses called realm. A realm is used to represent variability in a product family. For the single discriminant, a product member can be specified with a subclass in a realm. The multiple discriminants also represent an inheritance hierarchy that consists of a base class and realm. However, a product member can be specified with one or more subclasses in a realm. The optional discriminant is represented by two classes with a 0..1 association. A product member, which has been specified with a class, may or may not be specified with another class.

2.2.3. Feature Modeling

This technique was initially proposed in FODA [Kang, K., at el. 1990] to assist the activity of domain analysis. Many approaches apply and extend the definition of a feature model to support the development of software product line. However, feature modelling has not yet been standardised comparing with UML modelling which standard has been known. We describe below different aspects of feature modeling technique that are applied in existing approaches i.e.

(i) types of a feature in a feature model,
(ii) notation, and
(iii) relationships between features in a feature model.

In general, there are three types of feature: mandatory feature [Bosch, J. 1998][Clements, P., and L. Northrop. 2004][Griss, M. L., at el. 1998][Kang, K., at el. 1990][Kang, K. C., at el. 1998] is compulsory for product members of a family; optional feature [Bosch, J. 1998][Clements, P., and L. Northrop. 2004][Griss, M. L., at el. 1998][Kang, K., at el. 1990][Kang, K. C., at el. 1998][Svahnberg, M., at el. 2001] may exist in a specific product member or not; and alternative feature [Bosch, J. 1998][Clements, P., and L. Northrop. 2004][Kang, K., at el. 1990][Kang, K. C., at el. 1998] or variant feature [Griss, M. L., at el. 1998], is a possible feature that can be selected for a specific product member. Moreover, [Svahnberg, M., at el. 2001] define an extra type of feature external feature as a requirement not available in the system but need to be satisfied by the external system. A feature may be depicted as a round or a rectangle with its name inside. Some approaches have applied UML notation for expressing features [Griss, M. L., at el. 1998]. Moreover, different types of a feature i.e. mandatory, optional, and alternative are represented in different notations.

Regarding the relationships between features in a feature model, ideally, features are atomic units that can be put together in a product without difficulty. However, features are generally not independent and several types of relations can exist between them. According to [Gibson, P., at el. 1997], feature interaction is defined as a characteristic of a system whose complete behaviour does not satisfy the separate specifications of all its features. The types of relationships express the rules of feature interaction. These relationships are considered for selecting features. They represent which features must be selected together and which features must not.

A feature model becomes a powerful, practical, extensible, and simple technique in domain analysis process. On the other hand, UML diagrams, particularly class diagram, has been applied in domain design process due to its maturity, compliance; and practicality. We investigate how to map a feature model into UML class diagram as the following section.

2.3. MAPPING DIFFERENT PERSPECTIVES BETWEEN FEATURE MODEL AND UML DIAGRAM

According to several different perspectives of a software product line, the commonality and variability of the software product line are represented in multi-viewpoints. We focus on a feature model and UML diagram. We also investigate how the commonality and variability are captured by using feature-based and UML components.

2.3.1. Commonality

The commonality specified in a feature model is represented by mandatory features while the commonality specified in a UML class diagram is represented by a parent class having a composition association with child class. As shown in Figure 2.1, mandatory features top, body, door, and window must be specified in any car while class car is composed of classes top, body, door, and window.

2.3.2. Variability

As the variability specified in a feature model is represented by alternative and optional features, there are several possible ways to represent the variability in a UML class diagram. We describe below the mapping between optional and alternative features of a feature model into an UML class diagram.

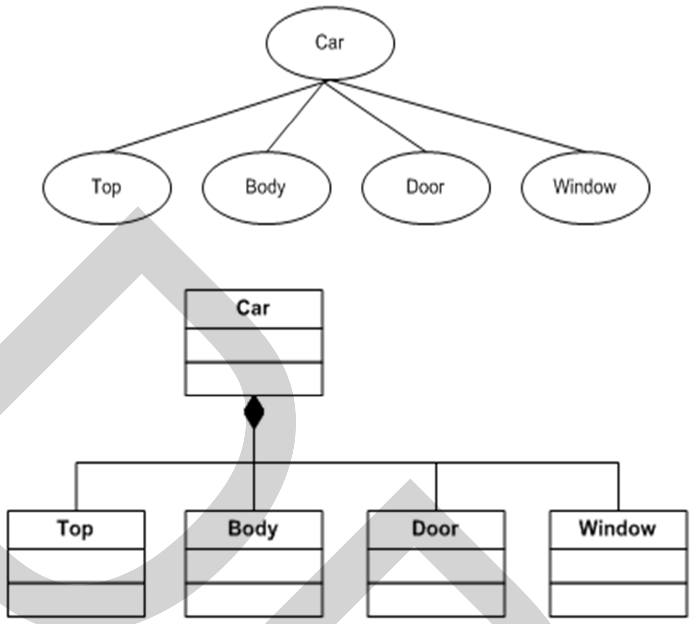
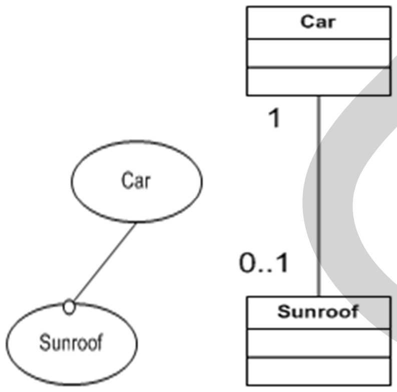
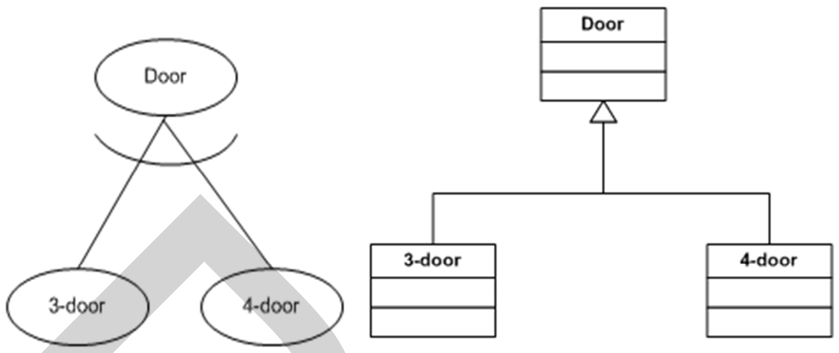


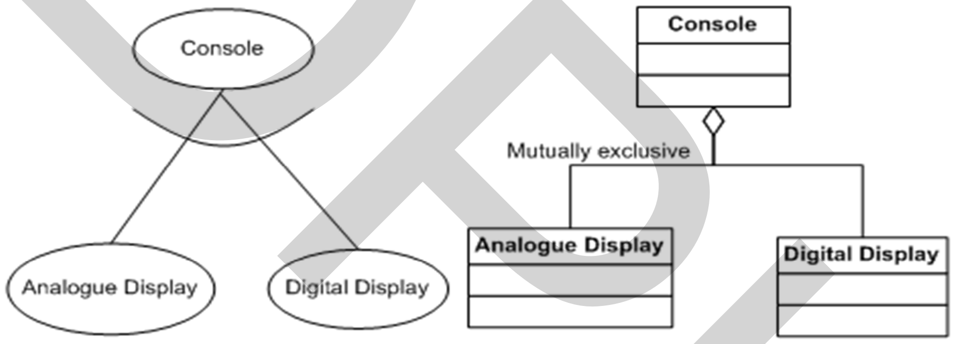
Figure 2.1: representing commonality in a feature model and UML class diagram



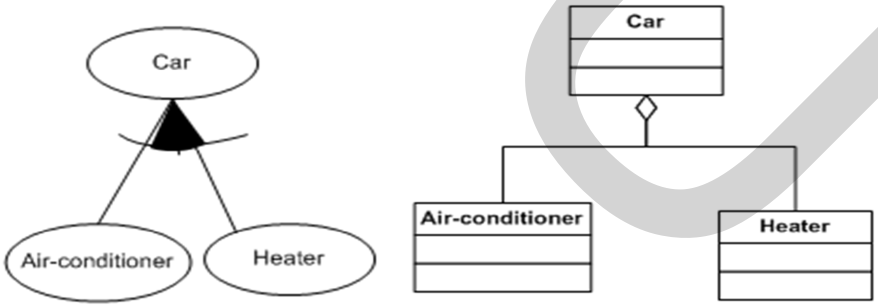
(a)



(b)



(c)



(d)

Figure 2.2: representing variability in a feature model and UML class diagram

Optional feature:

In a UML class diagram, it can be specified by a parent class having a binary association with another class with cardinality (0..1). As shown in Figure 2.2(a), an optional features sunroof may be chosen for a specific car while an instance of class car may be associated with an instance of class sunroof

Alternative feature:

It can be differently represented in a UML class diagram.

(i) one of alternative features must be chosen for a specific product member. There are two possible ways to represent this kind of variability in a UML class diagram. Firstly, as shown in Figure 2.2(b), alternative features *3-door* and *4-door* can be specified for a particular car while a class car can be specialized as either a class *3-door* or class *4-door*. Secondly, alternative features can be captured with a mutually exclusive association in UML diagram. As shown in Figure 2.2(c), a *console* can be either *analogue* or *digital display*.

(ii) one or more of alternative features can be chosen for a specific product member. As shown in Figure 2.2(d), a particular *car* can have either feature *air-conditioner* or *heater*, or both while a class car can be composed of a class *air-conditioner* or class *heater*, or of both classes.

(iii) Zero or more of alternative features can be chosen for a specific product member. There are two possible ways to represent this kind of variability in a UML class diagram. Firstly, classes and associations in a UML class diagram which capture this kind of variability can be represented by using a binary association with cardinality (0..1). Secondly, it is represented by a parent class having a composition association with children classes with cardinality (0..1). As shown in Figure 2.3, optional features *LCD screen*, *CD player, cassette player,* and *massage seat* may be specified in any car while class car may be composed of classes *LCD screen, CD player, cassette player,* and *massage seat*.

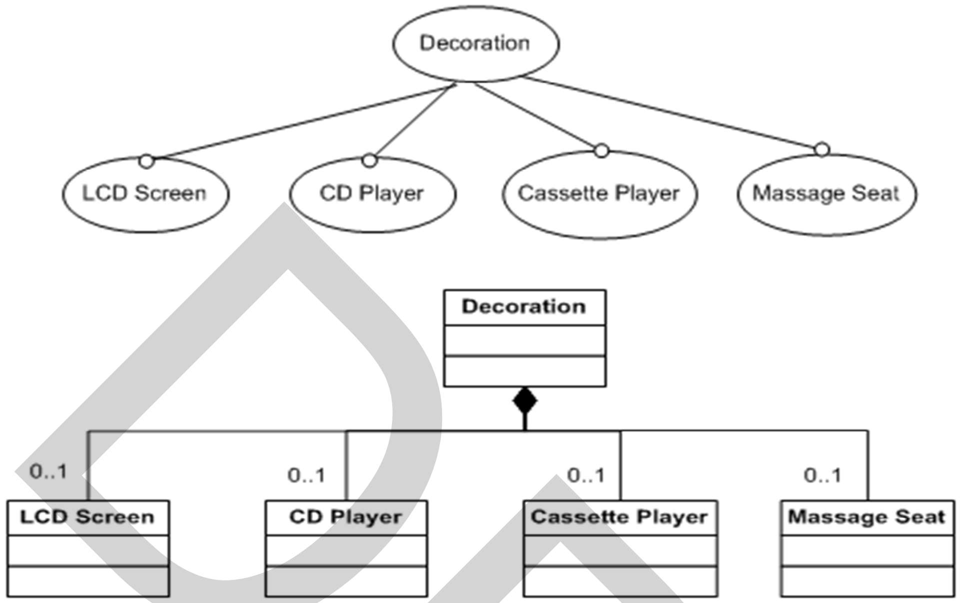


Figure 2.3: representing variability in a feature model and UML class diagram

In addition, it is possible that a feature is transformed to be an operation or attribute of a class in a class diagram. For example, a feature *colour* appears as an attribute of a class *body*. Moreover, a feature can appear as a composition of class(es), attribute(s), and/or operation(s). For example, a feature *maximum_speed* can be captured in an UML class diagram by an attribute *accelerating_power* and *operation speed*. The two aspects is a composite representation of a feature *maximum_speed*.

2.4. Waterfall Approach

There are various software development approaches defined and designed which are used/employed during development process of software, these approaches are also referred as "Software Development Process Models". Each process model follows a particular life cycle in order to ensure success in process of software development.

Waterfall approach was the first process model to be introduced and followed widely in Software Engineering to ensure success of the project. In "The Waterfall" approach, the whole process of software development is divided into separate process phases.

The phases in Waterfall model are:

(i) Requirement Specifications phase,

(ii) Software Design,

(iii) Implementation and

(iv) Testing & Maintenance.

All these phases are cascaded to each other so that second phase is started as and when defined set of goals are achieved for first phase and it is signed off, so the name "Waterfall Model". All the methods and processes undertaken in Waterfall Model are more visible.

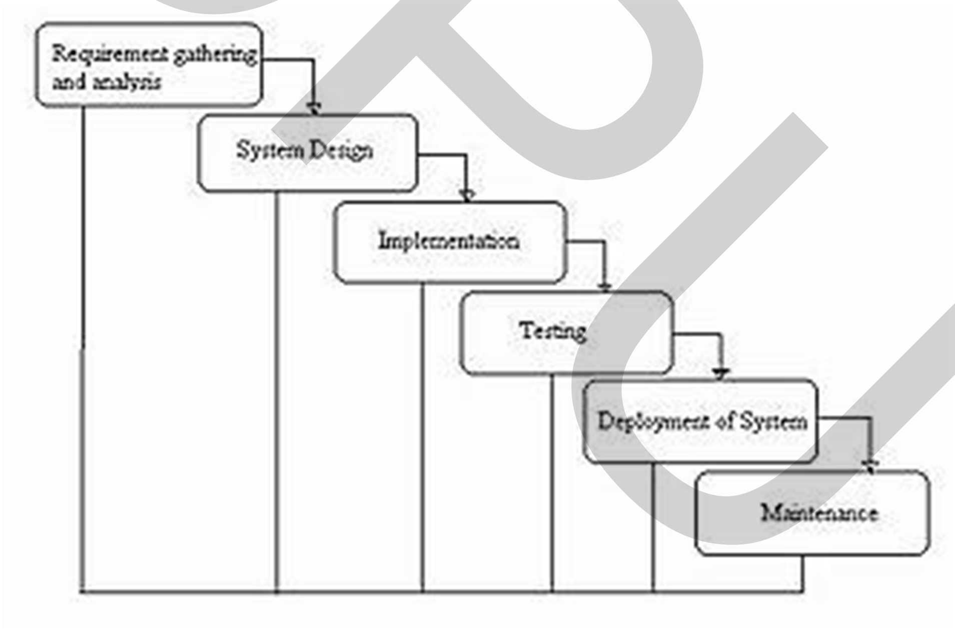


Figure 2.4: General overview of waterfall model

2.4.1. The Stages of "The Waterfall Model"

I. Requirement Analysis & Definition:

All possible requirements of the system to be developed are captured in this phase. Requirements are set of functionalities and constraints that the end-user, who will be using the system, expects from the system. The requirements are gathered from the end-user by consultation, these requirements are analyzed for their validity and the possibility of incorporating the requirements in the system to be developed is also studied. Finally, a Requirement Specification document is created which serves the purpose of guideline for the next phase of the model.

II. System & Software Design:

Before a starting for actual coding, it is highly important to understand what we are going to create and what it should look like. The requirement specifications from first phase are studied in this phase and system design is prepared. System Design helps in specifying hardware and system requirements and also helps in defining overall system architecture. The system design specifications serve as input for the next phase of the model.

III. Implementation & Unit Testing:

On receiving system design documents, the work is divided in modules/units and actual coding is started. The system is first developed in small programs called units, which are integrated in the next phase. Each unit is developed and tested for its functionality; this is referred to as Unit Testing. Unit testing mainly verifies if the modules/units meet their specifications.

IV. Integration & System Testing:

As specified above, the system is first divided in units which are developed and tested for their functionalities. These units are integrated into a complete system during Integration phase and tested to check if all modules/units coordinate between each other and the system as a whole behaves as per the specifications. After successfully testing the software, it is delivered to the customer.

V. Operations & Maintenance:

This phase of "The Waterfall Model" is virtually never ending phase. Generally, problems with the system developed, which are not found during the development life cycle, come up after its practical use starts, so the issues related to the system are solved after deployment of the system. Not all the problems come in picture directly but they arise time to time and needs to be solved; hence this process is referred as Maintenance.

2.4.2. Implementation with Waterfall Approach

The main characteristic of waterfall development is that it allows for departmentalization and managerial control. A schedule can be set with deadlines for each stage of development and a product can proceed through the development process like a car in a carwash, and theoretically, be delivered on time. Development moves from concept, through design, implementation, testing, installation, troubleshooting, and ends up at operation and maintenance. Each phase of development proceeds in strict order, without any overlapping or iterative steps.

However, waterfall development is that it does not allow for much reflection or revision. Once an application is in the testing stage, it is very difficult to go back and change something that was not well-thought out in the concept stage. Alternatives to the waterfall model include joint application development (JAD), rapid application development (RAD), synch and stabilize, build and fix, and the spiral model.

2.4.3. Common Errors in Requirements Analysis

In the traditional waterfall model of software development, the first phase of requirements analysis is also the most important one. This is the phase which involves gathering information about the customer's needs and defining, in the clearest possible terms, the problem that the product is expected to solve.

This analysis includes understanding the customer's business context and constraints, the functions the product must perform, the performance levels it must adhere to, and the external systems it must be compatible with. Techniques used to obtain this understanding include customer interviews, use cases, and "shopping lists" of software features. The results of the analysis are typically captured in a formal requirements specification, which serves as input to the next step.

In reality, there are a number of problems with this theoretical model, and these can cause delays and knock-on errors in the rest of the process. This article discusses some of the more common problems that project managers experience during this phase, and suggests possible solutions.

Problem 1: Customers do not really know what they want

Possibly the most common problem in the requirements analysis phase is that customers have only a vague idea of what they need, and a software engineer should ask the right questions and perform the analysis necessary to turn this amorphous vision into a formally-documented software requirements specification that can, in turn, be used as the basis for both a project plan and an engineering architecture.

To solve this problem, it is suggested that a software engineer should:

- Ensure that the software engineer spend sufficient time at the start of the project on understanding the objectives, deliverables and scope of the project.
- Make visible any assumptions that the customer is using, and critically evaluate both the likely end-user benefits and risks of the project.
- Attempt to write a concrete vision statement for the project, which encompasses both the specific functions or user benefits it provides and the overall business problem it is expected to solve.
- Get a customer to read, think about and sign off on the completed software requirements specification, to align expectations and ensure that both parties have a clear understanding of the deliverable.

Problem 2: Requirements change during the course of the project

The second most common problem with software projects is that the requirements defined in the first phase change as the project progresses. This may occur because as development progresses and prototypes are developed, customers are able to more clearly see problems with the original plan and make necessary course corrections; it may also occur because changes in the external environment require reshaping of the original business problem and hence necessitates a different solution than the one originally proposed.

Good project managers are aware of these possibilities and typically already have backup plans in place to deal with these changes.

To solve this problem, it is suggested that a software engineer should:

- Have a clearly defined process for receiving, analyzing and incorporating change requests, and make a customer aware of his/her entry point into this process.
- Set milestones for each development phase beyond which certain changes are not permissible -- for example, disallowing major changes once a module reaches 75 percent completion.
- Ensure that change requests and approvals are clearly communicated to all stakeholders, together with their rationale, and that the master project plan is updated accordingly.

Problem 3: Customers have unreasonable timelines

It is quite common to hear a customer say something like "it is an emergency job and we need this project completed in X weeks". A common mistake is to agree to such timelines before actually performing a detailed analysis and understanding both of the scope of the project and the resources necessary to execute it. In accepting an unreasonable timeline without discussion, in fact, it is quite likely that the project will either get delayed because it was not possible to execute it in time. Or the project will suffer from quality defects because it was rushed through without proper inspection.

To solve this problem, it is suggested that a software engineer should:

- Convert the software requirements specification into a project plan, detailing tasks and resources needed at each stage and modeling best-case, middle-case and worst-case scenarios.
- Ensure that the project plan takes account of available resource constraints and keeps sufficient time for testing and quality inspection.
- Enter into a conversation about deadlines with a customer, using the figures in a draft plan as supporting evidence for statements. Assuming that a plan is reasonable, it's quite likely that the ensuing negotiation will be both productive and result in a favorable outcome for both parties.

Problem 4: Communication gaps exist between customers, engineers and project managers

Often, customers and engineers fail to communicate clearly with each other because they come from different worlds and do not understand technical terms in the same way. This can lead to confusion and severe miscommunication, and an important task of a project manager, especially during the requirements analysis phase, is to ensure that both parties have a precise understanding of the deliverable and the tasks needed to achieve it.

To solve this problem, it is suggested that a software engineer should:

- Take notes at every meeting and disseminate these throughout the project team.
- Be consistent in the use of words. Make a glossary of the terms that are used right at the start, ensure all stakeholders have a copy, and stick to them consistently.

Problem 5: The development team does not understand the politics of the customer's organization

The scholars Bolman and Deal suggest that an effective manager is one who views the organization as a "contested arena" and understands the importance of power, conflict, negotiation and coalitions. Such a manager is not only skilled at operational and functional tasks, but he or she also understands the importance of framing agendas for common purposes, building coalitions that are united in their perspective, and persuading resistant managers of the validity of a particular position.

These skills are critical when dealing with large projects in large organizations, as information is often fragmented and requirements analysis is hence stymied by problems of trust, internal conflicts of interest and information inefficiencies.

To solve this problem, it is suggested that a software engineer should:

- Review existing network and identify both the information needed and who is likely to have it.
- Cultivate allies, build relationships and think systematically about the social capital in the organization.
- Persuade opponents within a customer's organization by framing issues in a way that is relevant to their own experience.
- Use initial points of access/leverage to move an agenda forward.

2.5. Summary

This chapter has provided background of software product line and waterfall approach. The framework of software product line artefacts is presented and mapping different perspectives between feature model and UML diagrams is also discussed.

Chapter III Research Method

This chapter presents the research method applied in the research. The goal of this research is described in Section 3.1. and the description of empirical project development based on software product line process and waterfall process are provided in Sections 3.2. and 3.3. Moreover, the case of new requirements management is described in Section 3.4.

3.1. Introduction

The goal of this research is to compare the qualitative and quantitative aspects between software product line -based and waterfall-based development and maintenance. To achieve the goal, this research conducted an experiment involving three software development projects that have some similar and different requirements. A team of developers was required to achieve the software development projects two times:

(i) to follow the software product line process, and

(ii) to follow the conventional software process, specifically waterfall process.

3.2. Empirical Project Development based on Software Product Line Process

The project started with developers being trained about software product line process and its 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.

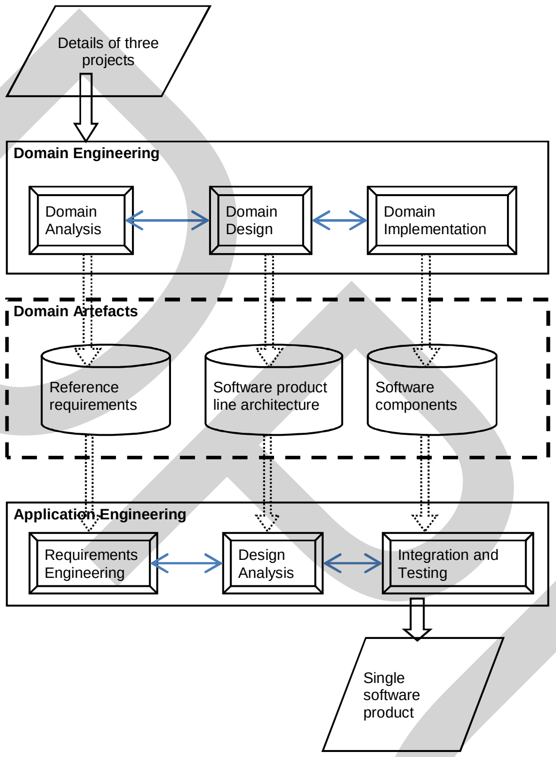


Figure 3.1. Software Produce Line Process

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 3.1.

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.

3.3. Empirical Project Development based on Conventional Software Process

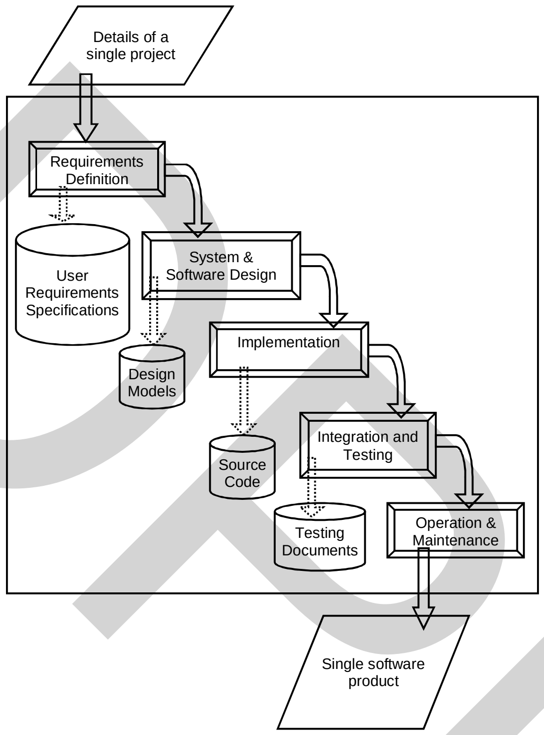


Figure 3.2. Waterfall Process

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 single 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.

The project that used waterfall-based model produced more artefacts than that of software product line process does. However, the development time of the waterfall-based project is greater than that of software product line. All the end of this step, we calculated and analyzed the qualitative and quantitative aspects in each project.

3.4. New Requirements Management on Software Products

In this phase, the team of developers was given new requirements on the systems. Many factors lead into this scenario, for example, customers require new functionality to be done in a design part of a software product.

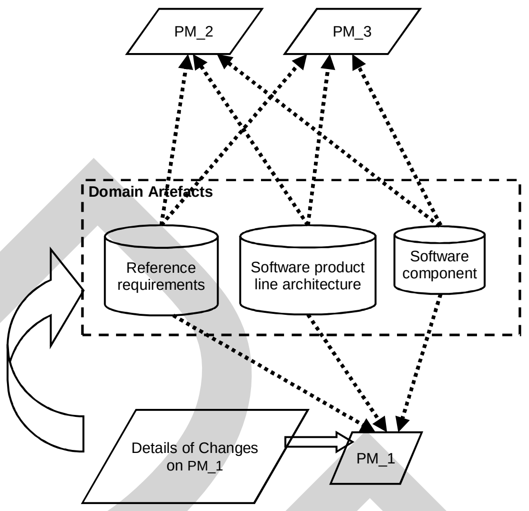


Figure 3.3. Maintenance on a product member of software product line

For the software product line-based systems, it is supposed the situation in which the organisation 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 3.3.

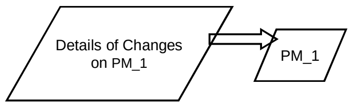


Figure 3.4. Maintenance on a software product

For the waterfall-based systems, it is supposed the situation in which the organisation has individually developed a set of software systems. Those are created from the development phase. And the new requirements are done to a software product. Therefore, it is necessary to evaluate only how these new requirements will affect any artefacts of the software product as shown in Figure 3.4.

3.5. Summary

This chapter has described the research method that is applied to the research. In next chapter, we present the experiments developed to demonstrate the work and analyses the experimental results of applying with both software development approaches.

Chapter IV Experiments and Results

This chapter provides the description of the experiments concerning with two software development approaches. The case study encompassing three software projects are presented. The motivation of different scenarios of software projects is described. Also, the experience and results of the case study are described.

4.1. Introduction to Experiments

The developers were requested to develop three software products, namely PM_1, PM_2, and PM_3. The list of functionalities and specifications of each software product is shown in Table 1. We describe below the details of each product.

PM_1

The software product PM_1 is expected to be a software system which supports an accounting department. And users prefer informative user interface. As shown in Table 4.1, the software product PM_1 has some basic functionalities:

(a) *processing accounting data*,

(b) *generating reports*,

(c) *managing log files*, and

(d) *displaying time and date*.

Additionally, it has some advanced functionalities:

(a) *access Internet* which allows a user to browse and download data through the Internet, and

(b) *email system* which supports the email.

PM_2

The software product PM_2 supports work related to accounting and offers a simple design and is targeted for users who are not familiar using a computer. The software product PM_2 has only basic functionalities:

(a) *processing accounting data*,

(b) *generating reports*,

(c) *managing log files*, and

(d) *displaying time and date*.

PM_3

The target customers of the software product PM_3 are users who work via the Internet. The system supports web-based environment. As shown in Table 4.1, the software product PM_3 has some basic functionalities:

(a) *processing accounting data*,

(b) *generating reports*,

(c) *managing log files*, and

(d) *displaying time and date*.

In addition, PM_3 offers advanced functionalities:

(a) *access Internet* which allows a user to browse and download data,

(b) *email system* which supports the email,

(c) *send and receive text messages,* and

(d) *send and receive multimedia messages*.

Table 4.1 shows the functionalities of each software product

Functionality	PM_1	PM_2	PM_3
F1	X	X	X
F2	X	X	X
F3	X	X	X
F4	X	X	X
F5	X		X
F6	X		X
F7	X		X
F8			X

F1: Processing accounting data

F2: Generating a report

F3: Managing log files

F4: Displaying time and date

F5: Enabling access the Internet

F6: Enabling emailing

F7: Sending and receiving text messages

F8: Sending and receiving multimedia message

Although the software products have some similar functionalities, the user requirements on each functionality are different. For example, the user interface, the template of report, and the details of log files. We firstly developed the software products, PM_1, PM_2, and PM_3, by applying software product line approach, and we then developed the software products by applying waterfall approaches. The details of development are described in the following sections.

4.2. The Development based on Software Product Line Approach

As described in Section 4.1., we applied software product line approach to develop the software products. According to the approach, the software products are recognized as software product members and the software product line architecture is created to support the members. Those software product members provide some similar and different functionalities and each one's functionalities are shown in Table 4.1.

Additionally, as described in chapter 2, there are many activities and difficulties associated with software product line engineering. Moreover, as proposed in [Krueger, C.W.], organisations can develop software product line systems in three different ways:

(a) proactive, when an organisation decides to analyse, design, and implement a line of products prior to the creation of individual product members;

(b) reactive, when an organisation enlarges the software product line system in an incremental way based on the demand of new product members or new requirements for existing products; and

(c) extractive, when an organisation creates a product line based on existing product members by identifying and using common and variable aspects of these products.

These approaches are not mutually exclusive and can be used in combination. For instance, it is possible to have a software product line system initially created in an extractive way to be incrementally enlarged over time by using a reactive approach. In addition, various stakeholders may be involved in the product line development process ranging from market researchers, to product managers, requirement engineers, product-line engineers, software analysts, and software developers. These stakeholders contribute in different ways to software product line engineering, have distinct perspectives of the system, and have distinct interests in different aspects of the product line. For example, a market researcher may be interested in the requirements and features of a new product member to be developed, while a software developer may be interested in the design and implementation aspects of this new product member. Therefore, the stakeholders would be interested in different types of documents and traceability relations that could assist them in their various tasks during system development.

In this research, we have conducted sets of experiments related to four different scenarios concerned with software product line engineering. More specifically, these scenarios include

(a) the creation of software product line,

(b) the creation of a new product member for an existing product line,

(c) changes to a product member in a product line system, and

(d) changes at the product line level.

These scenarios have been chosen since they illustrate the different ways in which organisations can develop software product line systems, as discussed above. For each of these scenarios we have identified the stakeholders involved in the process and the types of documents that are related to the scenarios.

4.2.1. Scenarios of Software Project Development based on Software Product Line Approach

We describe below each of the scenarios.

Scenario 1: Creation of a software product line

In this case, the stakeholders involved in this scenario are product managers that identify which aspects of the product members should be part of the software product line; and product line engineers, software analysts, and software developers that design and develop the documents at the product line level.

For this scenario, suppose the situation in which an organisation has no software product line and would like to create a product line that composes three members. In this case, all the domain analysis and design models of product members PM_1, PM_2, and PM_3 need to be compared in order to assist with identification of the information represented at the product line level.

Scenario 2: Creation of a new product member for an existing product line

This situation occurs when an organisation wants to enlarge its system and creates a new product member. In this case, traceability relations can be used to support the evolution of software systems and reuse of existing parts of the system. The stakeholders involved in this scenario are

(a) market researchers that are responsible for identifying the feasibility of creating a new product and the features that this new product should include from a commercial point-of-view;

(b) requirements engineers that specify the requirements of the new product;

(c) product line engineers that identify which aspects in the product line level are related to the new product;

(d) software analysts that analyse existing product members and identify the commonality and differences between existing product members and the new product; and

(e) software developers that design the new product by reusing parts of existing product members and specifying new aspects of the product being developed.

For this scenario, suppose the situation in which the software product line in an organization contains product member PM_2 and the organization wants to develop product member PM_1 from our case study. Consider that the requirements of PM_1 have been specified in four different use cases, as shown in Table 4.2. In order to be able to identify the similarities and differences between PM_1 and PM_2, the parts of PM_1 that can be reused from PM_2, and the parts of PM_1 that need to be developed, it is necessary to compare various documents including product line feature model, use cases of PM_1 and PM_2, and class, sequence, and statechart diagrams of PM_2. The types of documents to be compared and the relevant traceability relations associated with these documents.

Then, the set of use cases of PM_1 and PM_2 need to be compared with the feature model of the software product line in order to support the identification of similarities and differences between the use cases of PM_1 and PM_2. In addition, all class, sequence, and statechart diagrams of PM_2 are compared with the use cases of PM_1 to assist with the identification of which elements of PM_2 design models can be reused. It is also necessary to compare all class, sequence, and statechart diagrams of PM_2 with the use cases of PM_2 to assist with the identification of similarities and differences between the use cases of PM_1 and PM_2. Moreover, the class, sequence, and statechart diagrams of PM_2 need to be compared in order to support the identification of the elements that can be reused when designing PM_1.

Scenario 3: Changes to product members in a software product line system

In this scenario, stakeholders analyze of the implications of changes in the system. The stakeholders involved in this scenario are software analysts that specify changes to be made in a design part of a product member and, together with software developers, identify the effects of these changes in the other related design software artefacts.

For this scenario, supposed the situation in which an organisation has a software product line with product members PM_1, PM_2, and PM_3 from our case study, and that changes are made to the product members. Therefore, it is necessary to evaluate how these changes will affect the other design models of PM_1, PM_2, and PM_3. For example, if the changes on PM_1 affect the other product members in the same software product line, developers may consider how to manage the changes. The types of documents to be compared, for example, all the design models of PM_1 and PM_2 are compared in order to assist with the identification of information that may be affected by the changes.

Scenario 4: Changes at the software product line level

In this case, we investigate how to deal with the evolution and impact of the changes at the software product line level. More specifically, this scenario is concerned with changes at the product line level due to the addition of new features to the software product line system. The stakeholders involved in this scenario are market researchers that identify new features of the system and product line engineers that identify which aspects in the product line level are related to the new features and the effect of these new features to the other artefacts at the product line level. The types of documents to be compared.

4.2.2. Development Phase

During the development phase, Scenarios 1 and 2 are performed. More specifically, at the beginning the software product line has not been established, the developer team involves the establishment of it as planned in Scenario 1. Then, a new product member is added, the developer team involves the creation of the new product member into the software product line as planned in Scenario 2.

In particularly, the three software projects have been developed based on study, analysis, and discussions of business domain. Software systems are created based on demands which require a variety of software products. In this way, a number of documents are created by developers. 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.

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 [Jirapanthong, W. 2008]. The following artefacts are created:

(a) a feature model is created and composed of common features representing mandatory features, alternative and optional, representing different features between product members. For example, all product members must provide features of processing accounting data, generating a report, managing log files, and displaying time and date.

(b) a 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.

(c) seven process models are created and each is refined for a subsystem in the subsystem model.

(d) eleven 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. As below, the list of artefacts created for each product member is shown.

PM_1

(a) four use case descriptions
(b) a class diagram
(c) a statechart diagram
(d) four sequence diagrams
(e) source code

PM_2

(a) four use case descriptions
(b) a class diagram
(c) a statechart diagram
(d) four sequence diagrams
(e) source code

PM_3

(a) six use case descriptions
(b) a class diagram
(c) a statechart diagram
(d) six sequence diagrams
(e) source code

Table 4.2: Summary of number of document types used in the case study, number of main elements in the documents, and size of the documents

Document Type	**Number of Document Type**	**Element Type**	**Number of Element Type**
Feature Model	1	Features	130
Subsystem Model	1	Subsystems	5
Process Models	7	Processes	48 (total for all 7 process models)
Module Models	11	Modules	167 (total for all 11 module models)

Use Cases	PM_1 = 4
	PM_2 = 4
	PM_3 = 6
		Events	PM_1 = 37 (total for all 4 use cases)

			PM_2 = 36 (total for all 4 use cases)
			PM_3 = 44 (total for all 6 use cases)

Class Diagrams	PM_1 = 1
	PM_2 = 1
	PM_3 = 1
		Classes	PM_1 = 23

			PM_2 = 25
			PM_3 = 27
		Attributes	PM_1 = 26
			PM_2 = 26
			PM_3 = 33
		Methods	PM_1 = 78
			PM_2 = 82
			PM_3 = 87

Sequence Diagrams	PM_1 = 4
	PM_2 = 4
		Messages	PM_1 = 114 (in total for all 4 seq. diagrams)

	PM_3 = 6		PM_2 = 82 (in total for all 4 seq. diagrams)
			PM_3 = 112 (in total for all 6 seq. diagrams)
		Objects	PM_1 = 22 (in total for all 4 seq. diagrams)
			PM_2 = 21 (in total for all 4 seq. diagrams)
			PM_3 = 27 (in total for all 6 seq. diagrams)
Statechart Diagrams	PM_1 = 1 PM_2 = 1 PM_3 = 1	States	PM_1 = 4
			PM_2 = 4
			PM_3 = 4
		Transitions	PM_1 = 8
			PM_2 = 8
			PM_3 = 8

Table 4.2 shows a summary of the types and number of documents for each type, the number of elements in the documents.

4.2.3. Maintenance Phase

In this research, we also develop the case study of the maintenance phase. In particular, Scenarios 3 and 4 are performed. There are changes on product members in a software product line system and changes at the software product line level.

According to 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. In particular, the software artefacts are reusable and adaptable. A number of relations between artefacts are detected in order to determine the association between the new requirements and existing software artefacts in product members PM_1, PM_2, and PM_3 and software product line.

Different types of traceability relations are created to identify the role of those relations [8]. For example, the relations between the new requirements and software product line; between the new requirements and product member PM_1, and between software product line and product member PM_1. For instance, there are

(a) four use case documents for PM_1 and three processes in a process model of software product line that are related in terms of three different types of traceability relations (i.e. satisfies, implements, and refines);

(b) one class diagram and four sequence diagrams of software product line that are related in terms of containment. Those relations are then used in new requirements management process.

4.3. The Development based on Waterfall Approach

As described in Section 4.1., we also applied waterfall approach to develop the same software products. According to the approach, each software product is recognized as individual software projects, namely PM_1, PM_2, and PM_3. The functionalities of each software product are shown in Table 4.1.

As described in chapter 2, there are activities associated with waterfall software process. Various stakeholders may be involved in the software development process ranging from market researchers, to product managers, requirement engineers, software analysts, and software developers. These stakeholders contribute in different ways to software development, have distinct perspectives of the system, and have distinct interests in different aspects of software systems.

In this research, we have conducted sets of experiments related to four different scenarios concerned with software system development based on waterfall model. More specifically, these scenarios include

(a) the creation of software systems, and

(b) changes to software systems.

For each of these scenarios we have identified the stakeholders involved in the process and the types of documents that are related to the scenarios.

4.3.1. Scenarios of Software Project Development based on Waterfall Approach

We describe below each of the scenarios.

Scenario 1: Creation of software systems

In this case, the stakeholders involved in this scenario are product managers that identify the requirements of each software product; and product line engineers, software analysts, and software developers that design and develop the documents for the software systems.

For this scenario, suppose the situation in which the software systems are not available in the organization. In this case, all the analysis and design models of software systems need to be created. The stakeholders involved in this scenario are

(a) market researchers that are responsible for identifying the feasibility of creating a new software product and the features that this new product should include from a commercial point-of-view;

(b) requirements engineers that specify the requirements of the new product;

(c) software analysts that design the design models for new product; and

(d) software developers that implement the new product.

For this scenario, a development team does not need to consider other existing software systems. A set of documents i.e. use cases, class diagrams, statechart diagrams, and sequence diagrams of each software systems are created.

Scenario 2: Changes to software systems

In this scenario, stakeholders analyze of the implications of changes in the system. The stakeholders involved in this scenario are software analysts that specify changes to be made in a design part of a software product and, together with software developers, identify the effects of these changes in the other related design software artefacts.

For this scenario, supposed the situation in which the software systems PM_1, PM_2, and PM_3 are available, and that changes are made to those systems. Therefore, it is necessary to evaluate how these changes will affect each design models of PM_1, PM_2, and PM_3.

4.3.2. Development Phase

Similarly, the projects have been developed based on study, analysis, and discussions of business domain. The developers are required to reproduce the software systems based on the same set of requirements. Otherwise, this time they followed the waterfall software process model. According to the waterfall model, a number of artefacts for each single software product are created during software development process. As below, the artefacts of each single software product are checked and submitted to the repository.

PM1

(a) a usecase diagram
(b) four use case descriptions
(c) a class diagram
(d) a statechart diagram
(e) four sequence diagram
(f) source code

PM2

(a) a usecase diagram
(b) four use case descriptions
(c) a class diagram
(d) a statechart diagram
(e) four sequence diagram
(f) source code

PM3

(a) a usecase diagram

(b) four use case descriptions

(c) a class diagram

(d) a statechart diagram

(e) four sequence diagram

(f) source code

Table 4.3: Summary of document types, number of documents, element type, and number of elements in the documents, which are created for software project 1(PM_1)

Document Type	**Number of Documents**	**Element Type**	**Number of Elements**
Use Cases	6	Events	48 (total for all 6 use cases)
Class Diagrams	1	Classes	33
		Attributes	37
		Methods	98
Sequence Diagrams	6	Messages	165 (in total for all 6 seq. diagrams)
		Objects	32 (in total for all 6 seq. diagrams)
Statechart Diagrams	1	States	4
		Transitions	8

Table 4.4: Summary of document types, number of documents, element type, and number of elements in the documents, which are created for software project 2 (PM_2)

Document Type	**Number of Documents**	**Element Type**	**Number of Elements**
Use Cases	6	Events	46 (total for all 6 use cases)
Class Diagrams	1	Classes	35
		Attributes	36
		Methods	112
Sequence Diagrams	6	Messages	114 (in total for all 6 seq. diagrams)
		Objects	34 (in total for all 6 seq. diagrams)
Statechart Diagrams	1	States	4
		Transitions	8

Table 4.5: Summary of document types, number of documents, element type, and number of elements in the documents, which are created for software project 3(PM_3)

Document Type	**Number of Documents**	**Element Type**	**Number of Elements**
Use Cases	7	Events	61 (total for all 7 use cases)
Class Diagrams	1	Classes	37
		Attributes	43
		Methods	101
Sequence Diagrams	7	Messages	132 (in total for all 7 seq. diagrams)
		Objects	37 (in total for all 7 seq. diagrams)
Statechart	1	States	4

Diagrams		Transitions	8

Tables 4.3, 4.4, and 4.5 show the summary of the types and number of documents for each type, the number of various elements in the documents.

4.3.3. Maintenance Phase

For new requirements management on waterfall-based systems, developers divided their work based on their roles. Firstly, the developers summarized all new requirements from customers and reproduced new user requirement specification. Next, they 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. Finally, the developer delivered the customers the complete software when all of these stages finished.

4.4. Experience on Requirements Engineering and Change Management

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. Therefore several requirements modeling artefacts are applied. Table 4.6 summarises common artefacts for modeling requirements in projects.

Table 4.6. 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.

4.4.1. Experience on Requirement Development for Software Projects based on Software Product Line and Waterfall 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.

According to several types of requirements artefacts as shown in Table 4.6, the specification of requirements are done in different documents. 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 [Jirapanthong, W. 2008]. 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 single software development, a number of artefacts for each single software product are created during software development process. The artefacts of each single software product are usecase diagram, use case descriptions, class diagrams, statechart diagrams, sequence diagrams, and source code.

Moreover, there are several techniques for eliciting requirements, summarized in Table 4.7.

Table 4.7. 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	- Interviews must be schedules in advance - Interviewing skills are difficult to learn

		they would not publicly tell	
Reading	A wealth of written information available from which developers can discern potential requiremetns 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 writtern down - There are limits how much developers can read, and comprehend the information

4.4.2. Experience on Change Management for Software Projects based on Software Product Line and Waterfall Approach

According to 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 [Jirapanthong, W., A. Zisman. 2009].

For the software product line-based systems, it is supposed the situation in which the organisation 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 3.3.

For the single 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. Finally, the developer delivered the customers the complete software when all of these stages finished.

4.5. Analysis of Experiment Results

In this section, we analyse and evaluate the experiments by focusing on two aspects of measurement:

(a) qualitative and

(b) quantitative measurement.

4.5.1. Qualitative Measurement

In general, qualitative methods and tools for system analysis can address the problem of how to empirically determine the context of software process. In this research, we focused on comparison between two software process methodologies how they are practiced. As mentioned, we have conducted the survey and interview. It has been observed that the customers are satisfied with the software product line resulting projects and teamwork. Moreover, the software product line developers satisfied the process that emphasis the software more than the documentation. However, it has been also noticed that it is easier to train waterfall-based practices to inexperience developers but some experience developers tend to resist some software product line practices because

(a) they have to change their style in working, and

(b) it costs them for establishing the software product line artefacts.

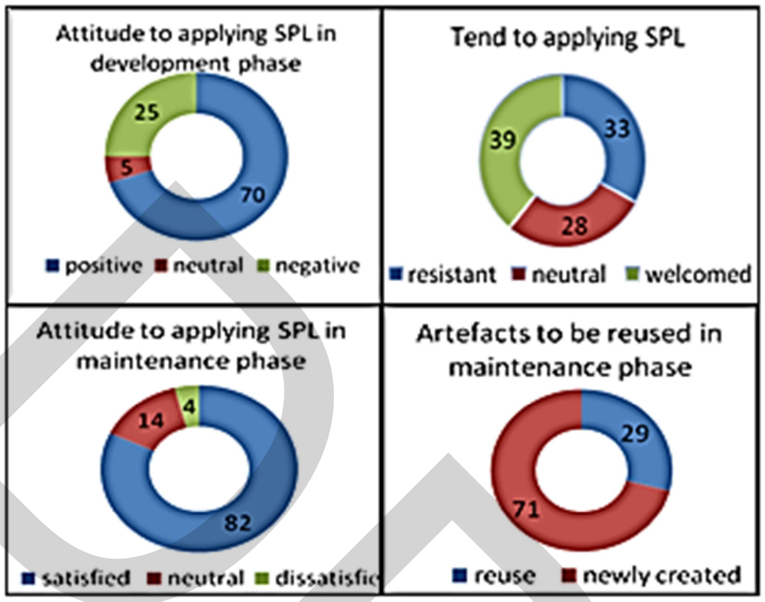


Figure 4.1. Qualitative measurement

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, the waterfall-based developers are unsatisfied to frequently update the documentation.

4.5.2. Quantitative Measurement

Basically quantitative metrics are fundamentally limited to the measurement of the size of system, time and effort spent during software development process. In this research, we measured the total of work hour spent during development and maintenance phases as well as the errors during the phases by following the software processes. In particular, we take account into the number of items causing a software system false. As mentioned earlier, the developer team was required to develop a set of three software products two times. One is applied with software product line and the other is applied with waterfall process.

As shown in Table 4.8, the result shows that the effort of software product line-based projects is less than waterfall-based 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. The more software artefacts are reused, the less time is spent. Although, developers spent extra time and effort to establish domain artefacts, it seems the trend of effort for new products in the same product line would decrease. On the other hand, for the waterfall-based projects, 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.

Table 4.8 shows the effort and errors during development phase

	Product Name	Work-hour	Error
Domain Engineering	-	620	22
SPL-based project1	PM_1	315	17
SPL-based project2	PM_2	240	15
SPL-based project3	PM_3	215	15
Waterfall-based project1	PM_1	765	28
Waterfall-based project2	PM_2	848	21
Waterfall-based project3	PM_3	684	14

However, the number of errors which occur during the development phase of software product line is high. Also, some defects are discovered during the integration process for a product member. It took some effort to fix them. Comparing with waterfall-based projects, there is less number of errors during development phase. It is because the developer team is well experienced on waterfall process than the other one.

For maintenance phase, we measured the total of time to achieve the new requirements as shown in Table 4.9. The result shows that the spending time of software product line -based projects is less than of waterfall-based projects. Developers who performed the maintenance phase found that well documentation can be useful and reduce the cost to complete the task. In particular, the artefacts of a waterfall-based project are more documentation than a software product line-based project. Otherwise, entire documentation of waterfall process is inaccessible to maintainers whereas documentation of software product line process is restored as repository to support in maintenance and reuse process.

Table 4.9 shows the effort and errors during maintenance phase

	Product Name	Work hour	Error
Software product line -based project1	PM_1	305	18
Waterfall-based project1	PM_1	380	17
Software product line -based project2	PM_2	25	11
Waterfall-based project2	PM_2	48	9
Software product line -based project3	PM_3	8	4
Waterfall-based project3	PM_3	23	6

4.6. Summary

This chapter has presented the experiments concerning with two software development approaches. The case study encompassing three software projects are presented. Also, the experience and results of the case study are described.

Chapter V Conclusions and Future Work

This chapter provides the conclusions, some useful suggestions for future study and, future work of this research. Section 5.1 presents the overall conclusions. The future work are described in Section 5.2.

5.1 Conclusions

In fact, the combination of different techniques and approaches is found in some works [CAFE. 2003][ESAPS][Gomaa, H., M. E. Shin. 2004][Jirapanthong, W. 2005], the author presented the framework of software product line artefacts which support activities in software product line development. The mapping between key software artefacts, particularly feature model and UML class diagram, was presented. Moreover, there are some situations that require the evolution of software product line such as:

(i) there is a change on existing product family; and

(ii) the reusable components of a product family have missed some functionalities.

Those situations occur when the maturity level of software product line in an organization has grown. The organisation requires a software process which implements new requirements and maintains the consistency of existing systems. An approach to evolve software product line should be investigated in order to enforce a standardised approach for evolution.

To conclude, we evaluated comparative study between software product line-based process and waterfall-based process. The productivity during development using software product line is higher than that using waterfall-based model. Also, a software product line-based project is more maintainable than waterfall-based one. 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.

Additionally, 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 single 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. 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 research, 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.

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, is clearly 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 single 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.

5.2 Future Work

The following issues are interesting directions for future work:

1. Automatic process

   At present, the model is applied with using several software tools, depending on its availability. The activities in software development process are performed in a semi-automatic way. More specifically, some activities are done by applying with software tools and some are manually performed. The automatic process is expected to support the activities of software process such as requirements elicitation and specification, requirements transformation into design, and implementation into coding.

2. Standardized approach for software product line evolution

   An approach to evolve software product line should be investigated in order to enforce a standardized approach for evolution.

3. Extension to small and medium-sized software projects

   The techniques and approaches for software product line development should be further extended to allow establishing software product line for small- and medium- sized projects. In the future work, we plan to gather more data from the projects in order to develop statistic evaluation of comparison between small or medium-sized and large-sized software projects.

References

Bosch, J. 1998. Product-Line Architectures in Industry: A Case Study. Pages 544 - 554. The 21st International Conference on Software Engineering. IEEE, Los Angeles, USA.

CAFE. 2003. from http://www.esi.es/en/projects/cafe/cafe.html.

Clauss, M. 2001. Modelling variability with UML. GCSE 2001 - Young Researchers Workshop.

Clements, P., and L. Northrop. 2002. Software Product Lines: Practices and Patterns. Addison-Wesley, Boston, MA.

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

ESAPS. From http://www.esi.es/en/Projects/esaps/esaps.html.

Fantechi, A., S. Gnesi, G. Lami, and E. Nesti. 2004. A Methodology for the Derivation and Verification of Use Casees for Product Lines. Pages 255-264. The 3rd International Conference, SPLC 2004. Springer Verlag, Boston, MA, USA.

Garlan, D. and M. Shaw. 1993. An introduction to software architecture. In Advances in Software Engineering and Knowledge Engineering, Volume I. World Scientific Publishing Company.

Gibson, P., B. Mermet, and D. Méry. 1997. Feature Interactions: A Mixed Semantic Model Approach in O'Regan and Flynn, eds. 1st Irish Workshop on Formal Methods (IWFM97), Dublin.

Gomaa, H., and M. E. Shin. 2004. A Multiple-View Meta-modeling Approach for Variability Management in Software Product Lines. Pages 274-285. 8th International Conference (ICSR 2004). Springer Verlag, Madrid, Spain.

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.

Jirapanthong, W. 2005. Supporting Product Line Development through Traceability. 12th Asia-Pacific Software Engineering Conference (APSEC 2005), Taipei, Taiwan.

Jirapanthong, W. 2008. *An Approach to Software Artefact Specification for Supporting Product Line System*s. the 2008 International Conference on Software Engineering Research and Practice (SERP'08), Las Vegas, Nevada, USA, 2008.

Jirapanthong, W., and A. Zisman. 2009. *XTraQue: traceability for product line systems*. Software and System Modeling 8(1): 117-144 (2009).

John, I., and D. Muthig. 2002. Tailoring Use Cases for Product Line Modeling. REPL'02, Essen, Germany.

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.

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

Krueger C.W. "Software Mass Customization",http://www.biglever.com/papers/BigLeverMassCustomization.pdf

Northrop, L. M. 2002. SEI's Software Product Line Tenets. IEEE Software 19: 32-40.

Svahnberg, M., J. Gurp, and J. Bosch. 2001. On the Notion of Variability in Software Product Lines. Pages 45-55. The Working IEEE/IFIP Conference on Software Architecture.

Biography

Name	Dr. Waraporn Jirapanthong
Education Background	• PhD. in Computer Science, Software Engineering Group, City University, London, UK. • MSc. in Computer Science, Faculty of Science, Mahidol University, Thailand. • BSc. in Computer Science (First Class Honors), Faculty of Science, Thammasat University, Thailand.
Employment	Assistant Professor , Faculty of Information Technology, Dhurakij Pundit University