Towards Understanding Requirement Evolution in a Software Product Line

An Industrial Case Study

Yijian Wu[1], Didar Zowghi[2], Xin Peng[1], Wenyun Zhao[1]
[1]School of Computer Science, Fudan University, Shanghai 201203, China
{wuyijian, pengxin, wyzhao}@fudan.edu.cn
[2]Faculty of Engineering and Information Technology, University of Technology, Sydney, Australia
didar.zowghi@uts.edu.au

***Abstract*—In most software development practices, software requirements and architecture are addressed simultaneously. The architecture may grow based on a core specification of requirements, and the requirements may also be elicited incrementally as the architecture gets more and more concrete. In this paper, we present a case study on the development history of Wingsoft Examination System Product Line (WES-PL), an active, industrial software product line with a history of more than eight years. We focus on 10 member products, 51 major versions that have been delivered to customer or archived in the repository between December 2003 and May 2012, by tracing both requirement and architectural changes. We identify a requirement change classification from the viewpoint of architectural impact. We claim that software requirements are negotiated and may be guided by existing software architecture design, especially in the process of software product line development. Product strategy requirements play an important role in marketing requirement negotiation. We also find typical evidences showing that a product leader or architect has to make difficult decisions to keep a balance between marketing requirements from customer-side and software architectural design from his own side.**

***Keywords*-requirement evolution; architecture evolution; software product line; industrial case study**

I. INTRODUCTION

In most modern software development, requirement changes are inevitable [9]. Software architects have to evaluate the impacts of requirements changes on the architectural design and repeatedly answer the question "is the (architectural) consistency links preserved by the changes in requirements" [5]. Decision making in the Evolution of software architecture is usually difficult because performing changes to the architecture is not straightforward. Therefore, it is important to fully understand the requirement changes (and also the requirements themselves) to aid the evaluation on the merit of responding to a requirement change request with a change in the architecture.

In software product line (SPL) development, changes to the requirements are even more frequent. An SPL provides an architectural level reuse approach for product development within a business domain. As software reuse is one of the typical scenarios in co-evolution of requirements and design [4], it is natural to consider the synergetic relationship between SPL requirements and architecture design. In SPL development, requirements are often expanding, more complicated and interrelated than in conventional software development. Requirement change requests originate from various customers continuously. Meanwhile, the product line architecture (PLA) serves as a core reusable asset for all member products. The architect shall maintain conformance between the PLA and individual product architectures. The complex relationship between requirements and architectures in SPL development leads to severe difficulty in analyzing the impacts of various requirement changes to the PLA, and thus makes architecture evolution decision challenging for software architects.

These challenges have also been encountered within the Wingsoft Company, a typical small to medium enterprise (SME) in China. In the past ten years, Wingsoft has adopted a light-weight SPL methodology [2, 3] for its software development and has thus become a major software vendor in several business domains, including high-end financial management, publication management, and education and examination solutions.

In this paper, we present an in-depth investigation into the development history of Wingsoft Examination Systems Product Line (WES-PL). We examine the collection of both requirements changes and architecture evolutions in 10 member products for several types of online examinations, including 51 major released or archived versions. We attempt to address the highly complex interrelationship between SPL requirements, PLA, and product architectures. Our case study differs from previous research on evolution [7,11,12], co-evolution [4,5,6,10], and case studies on industrial SPLs [7,12] in two ways: 1) We focus on understanding the mutual relationship between requirement and architecture evolution during SPL development and 2) the target SPL has a relatively long evolution history and has

comparatively larger number of member products [3] thus providing a richer experience for analysis.

We believe that the case study presented in this paper will contribute to better understanding the co-evolution of requirements and architecture in SPL and will offer opportunities to construct further ideas on how to study the interrelationship between requirements and architectures in SPL development.

The rest of the paper is organized as follows. Section II shows the business background of the WES-PL and an overview of the architecture of Exam-PL which is the focus of this case study. Section III classifies requirement and architecture evolution types in SPL and proposes a preliminary understanding of the relationship of the evolution types. Section IV further discusses facts found in the development history of the SPL and tries to inspire further research questions on understanding the co-evolution of requirements and the architecture. Section V concludes our work.

II. BACKGROUND

A. The WES-PL and the Exam-PL

Wingsoft Examination System Product Line (WES-PL) consists of over 20 products related to computer-aided examinations and practices. The first product of WES-PL is an oral examination system for Shanghai Municipal Education and Examination Authority (SMEEA). The first release of the product was in December 2003. The product has served for Oral English Test in College Entrance Examination (an official examination before a high school student may enter a university or college) for nine years. Now, other sibling products derived from the WES-PL cover five processes in the whole lifecycle of an examination, including designing examination paper, previewing examination paper, conduct examination, scoring, and training. In order to conduct a deep investigation on the requirements and architecture evolution history, we focus on the core examination process, which is implemented in a subset of the WES-PL, the Exam-PL.

B. Examination Types and Exam Product Categories

Each examination product derived from Exam-PL contains a server-side application and a client-side application. In each examination room, one server-side application and tens of client-side applications are deployed. The server-side application is operated as a controller by an examiner, while each client-side application is operated by an examinee and provides a software environment for the examinee to take the exam.

Based on various examination requirements that are proposed by examination authorities and educational organizations, we had identified roughly four types of examinations: 1) Human-machine oral examination; 2) Human-human oral examination; 3) Written examination; and 4) Examinations that requires external operations. Each type of examination creates a category of examination products, i.e. SOLO-Oral, MULTI-Oral, ONLINE, and EXT, as listed in Table I. In each category, several products may exist. We collected 51 major versions of all examination products that either officially released to the customers or formally archived in the project repository. Although there were hundreds of minor versions in the evolution history, we do not take them into account in our case study. Among the 51 versions, about 1/4 (14) are experimental, which means the development team attempted new design or technology in these versions as a tentative experiment.

Usually, a product falls in one specific product category. If a customer requires cross-category examinations, she will have to order two products. One exception is that, if a customer raises a requirement that is reasonable and ahead of a proper PLA design, the team may build an experimental product that temporarily combines two types of examinations for that customer. For example, the product for Xiamen Examination Center began to cover both SOLO-Oral and part of ONLINE examinations since September 2007, as a response to the new requirement "oral examinations and listening comprehension examinations should be taken together". This experimental product finally became a running prototype of the future ONLINE products. In this example, the flexibility introduced in an experimental product is evaluated for further decisions of whether a new category of products is worthy being created or whether the architectural changes should be merged into the PLA.

TABLE I. CATEGORY OF EXAMINATION PRODUCTS IN EXAM-PL

Category	Type of Examination	Comment	Time of the First Product	# of Products	# of Versions	# of Experi. Versions
SOLO-Oral	Human-machine oral examinations.	The software shows or plays exam questions and the examinee answers the questions orally.	Dec. 2003	4	34	13
MULTI-Oral	Human-human oral examinations	The software automatically divides all examinees in groups and transfers audio within each group.	Nov. 2004	2	10	0
ONLINE	Written examinations	Example tasks: multiple choice, fill in blanks, and writing. The questions may or may not contain audio and/or video contents.	Oct. 2008	2	4	1
EXT	Examinations that requires external operations	The software displays a non-full-screen panel with time remaining and other information. Example tasks: Coding with Visual Studio, Word Processing with Word2003, etc.	Mar. 2011	2	3	0
Total				10	51	14

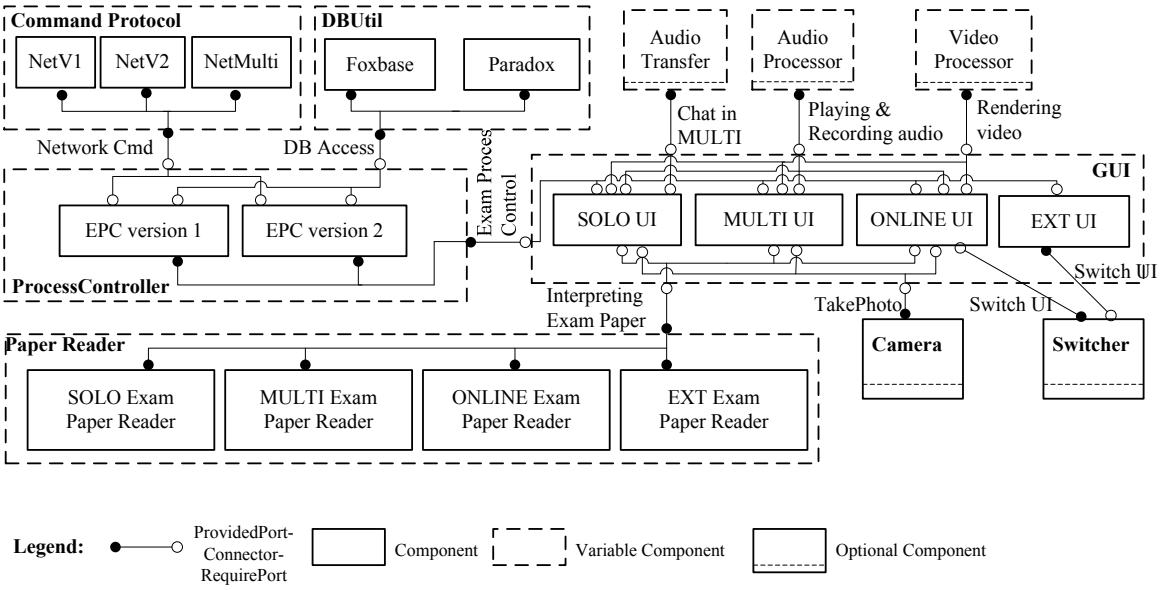

Figure 1. A sketch map of current Exam-PLA

C. A Sketch Map of the Exam-PL Architecture

To simplify the following discussion, we ignore trivial and non-changing architectural decisions (such as the Client/Server structure). We do not distinguish explicitly the software architecture of the Server Application and that of the Client Application, but depict an overall architecture skeleton as in Figure 1. One can see plenty of product variability in this figure although not all variant components are shown.

The architecture in Figure 1 shows a sketch map of current version of the PLA. Main components include:

Process Controller: This is a variant component that implements the process of an examination. There are typical stages in computer-aided examinations, such as Select Exam Papers, Send/Receive Exam Papers, Examinee Log-in, Examinee Confirming Personal Information, Audio Device Testing, Confirming Examination Notice, Taking Exam, Packaging Answers, and Collecting Answers. The process is controlled by ExamProcessController (EPC), which currently has two variants.

GUI: This is a graphic user interface component that implements all user interactions. Most of its interfaces are of type RequirePort. This component collects information from other functional components and displays it via graphic components. It is a variant component that supports different types of examinations.

Command Protocol: This is a network command component that encapsulates Client/Server communication definitions and operations. There are several different versions of Command Protocol for different examination types and for different network communication requirements.

DBUtil: This is a utility component that deals with database access. The variants deal with various database vendors.

Paper Reader: This is a component that reads examination paper files for various types of examinations and provides decoded data for GUI output.

Switcher: A newly added component that enables GUI switching from ONLINE UI to EXT UI and vice versa. ONLINE UI component initiates the switching and EXT UI is invoked upon request. It is an optional component that serves for ONLINE UI and EXT UI only.

Camera: This is an optional component that provides photographing functionality. It was introduced quite early (in year 2004) and is compatible with all products, but not used very much due to lack of requirements and lack of hardware supports, until a new requirement emerges in year 2012.

Audio Transfer: This is an optional component that transfers audio via network. It is bound only in MULTI-Oral Exam Products. It is also a variant component which has a Broadcast implementation and a Multicast implementation.

Audio Processor: This is an optional component that deals with audio playback and recording. It is bound only in products that need to play/record audios. It is also a variant component that supports different playback and recording implementations.

Video Processor: This is an optional component that deals with video rendering. It is bound only in products that need to render video clips. It is also a variant component that supports different rendering mechanisms.

In the following discussions, we will show evolution trace of both requirement and architecture and try to find common practices in requirement elicitation and negotiation in SPL development.

III. Requirement and Architecture Evolution

A. Requirement Change Types from the Viewpoint of Architectural Impact

There are several elements that can be used to classify requirement changes, such as types of change, reason of change, source of change, etc.[1] In SPL development, product strategic requirements are important and always guide the process. Therefore, requirements change requests are always evaluated by product manager or architect against the conformance to the product strategy and PLA. Some requirement change requests may be denied or postponed due to conflict with architectural design, while others may be accepted resulting in different architecture evolution decisions.

We have identified several types of requirement changes from the point of view of architectural impact as follows:

- Default requirement changes

Default requirement changes usually come from non-critical defects or bug reports. They usually have very small impact on the architecture because only particular defect in components are fixed or upgraded

- Urgent requirement changes

Urgent requirement changes are those that need urgent attention. They may come from a blocking bug report or an important bidding requirement. If a requirement change is labeled as "Urgent", the responding process should minimize the impact on the current architecture and fulfill the requirement in a timeframe as short as possible.

- Tentative requirement changes

Tentative requirement changes are usually internal requirements that may lead to new valuable features but may also bring uncertain risks to the PL.

- Customer-specific requirement changes

Customer-specific requirement changes usually com from a particular customer but are not representative in other customers. Most of them may not be considered as part of the domain requirements. Designs that fulfill such requirement changes are unlikely to be merged into the PLA.

- Large scale requirement changes

These requirement changes potentially have deep impact to current architecture (and so to future products within product strategy) but the changes are not urgent. These requirements are usually new and are proposed internally from within the company (such as a marketing department). In this situation, an architecture clone is usually made but the merge possibility is still undetermined. There might be a decision for the architect to make whether this request is a potential new product line or only a big change in current PLA.

- Internal maintainability requirement changes

These requirement changes describe the degree to which the development team is likely to improve the maintainability of the whole product line or all member products. This type of requirement changes usually emerges when the PLA and the application architectures evolve asynchronously to a degree to which maintenance begins to cost too much. Thus an architect may initiate an architectural decision process to synchronize among the PLA and all application architectures.

B. Architecture Evolution Types

In our previous work [3], we identified six typical architectural evolution types, namely *linear evolution*, *clone*, *derivation*, *merge*, *synchronization*, and *propagation*. A summary of these evolution types is shown in Table II.

These six architectural evolution types take place during the evolutions of the PLA and the architectures of all member products. In our previous case studies we discovered that in industrial product line development, especially in SMEs, the PLA and the application architectures may not be synchronously evolved. An architect may choose a reactive approach [2] to maintain each member product, only to save short-term effort and minimize risks [3].

We also confirm the conclusion in this case study with the Exam-PL. In the next subsection, we will present the relationship between Requirement change types and Architectural evolution types.

TABLE II. A SUMMARY OF ARCHITECTURE EVOLUTION TYPES

Evolution type	Architectural Change Flow	
	From	To
Linear evolution	An App. Architecture	**next version** of App. Architecture
	PLA	**next version** of PLA
Clone	An App. Architecture	**new** App. Architecture
Derivation	PLA	**new** App. Architecture
Merge	App. Architectures	**next version** of PLA
Synchronization	PLA	**all** existing App. Architectures
Propagation	An App. Architecture	**some** other existing App. Architecture(s)

C. Relationship between Requirement Change Types and Architectural Evolution Types

Architectural evolution is typically driven by requirement changes externally and/or internally. We find that the six architecture evolution types have various particular driving forces. *Linear*, *Clone*, and *Derivation* are driven by marketing (external) requirement changes. *Synchronization*, *Propagation*, and *Merge* are usually driven by internal requirement changes[1] which aim at improving the easiness of maintenance with in the development team.

Further, every architectural evolution has a possibility to be ultimately merged into the PLA. The to-be-merged possibility of an architectural evolution is highly related to the driving requirement changes. Therefore, we relate requirement change categories, architectural evolution types, and to-be-merged possibility in Table III, showing how we understand the impacts that requirement changes have on architecture evolution decisions.

[1] Propagation may also be driven by external requirement changes but we find this situation rare in development reality.

TABLE III. POSSIBLE ARCHITECTURAL IMPACTS OF REQUIREMENT CHANGES

Requirement change cat.	Architectural Solution	Result in	Merge possibility
Default	Linear	New version	High
Urgent	Clone (with local updates)	New branch	High
Tentative	Clone	New branch	Medium, depending on test results
Customer-specific	Clone	New branch	Low
Large scale	Clone	New branch	Undetermined
Internal	Synchronization or Propagation	New version	High

On the other hand, architecture design has also an impact on requirement elicitation and negotiation. In SPL development environment, it is possible that a requirement change request will be denied or postponed due to its conflict with the long-term strategy of the PL. In real industrial development, we find several cases that the architect and product manager has successfully persuaded the customer to yield their original change request. That is also part of the reason why the PLA has not been changed much during the past years. We will discuss this in more details in Section IV A.

IV. DISCUSSION

In this section, we discuss the mutual effects between architectural design and requirement elicitation in the Exam-PL evolution history. We ask two questions about this synergistic relationship and one question on the possibility of developing an approach to assist continuous evolution decisions on both requirement side and architecture side.

A. How does architectural design grow as requirements are being elicited?

1) The initial architecture is "lazy".

We found the growth of the initial architecture to be rather slow. Although there are more than 10 major versions during eight years, the architecture still looked much like the initial one (i.e. the one designed for SMEEA in the second half of the year 2003), after the basic requirements were stated. We discovered that, in the next eight years (till early in 2012), the architecture was not changed very much at all. Most of the changes were limited to part of the architecture, such as adding new components, replacing old components, local changes in component connections, and refactoring large components into smaller ones. The PLA was created based on the initial architecture. It was also stable on overall. The only one systematic refactoring to the PLA happened in 2008 [3]. Most of the changes were incorporating variations of different products.

The reason behind this relative stability of the architecture is possibly the product strategy requirements. The product is designed for general purpose of oral examinations. Therefore, whenever a market requirement emerges, the architect evaluates whether the new requirement meets the product strategy and then seeks an architectural solution that minimizes impacts to the current architecture. This design decision leads to a "lazy" architecture.

We believe that the "lazy" feature reflects the fact that domain requirements can be adapted to accommodate existing architectural design. As long as the maintenance effect is affordable, the architects tends to keep the architectural design by negotiating or denying some requirements. This does not mean that the original design is perfect, but the architecture is acceptable regarding current requirements or the current requirements are negotiable. This situation is to be discussed in Subsection B as an effect that architectural design has on requirements.

On the other hand, requirements with deep impacts on the current PLA may lead to a fork of current PLA into a new product line. The creation of a new product line (in the same product family) was observed in the development history of Wingsoft Examination System Product Family (WES-PF) where *Training* products formed a new product line as too many differences were found in new training requirements. Although the architectures are evolving separately, the new PL also shares some domain assets with the original PL since they are still within the same product family of a business domain. The new PL is also an example for the "lazy" feature because the creation of the new PLA helps in keeping the original PLA stable by isolating all major architectural changes into a new PLA.

2) Architectural design is iteratively refined for the purpose of flexibility as new market requirements emerge.

In our investigation in this case study, we observed that from time to time a component is split into several smaller components to provide more flexibility when new market requirements emerge.

This refinement process is similar to the usual way of refining components, but the differences are in that the purpose of the refinement is to prepare for variations in the architecture level and that it is not purely in the design phase but iteratively in design, implementation, and redesign phases during product line evolution.

In our case study, we encountered cases that a higher-level component that was originally supposed to be elaborated into several detailed components was actually implemented directly as a whole by a developer. The supposed-to-be lower-level components were not independently implemented but hard-coded together as an implementation of the higher-level component. This is probably because the lower-level designer and the implementer were the same person who did not see the necessity to elaborate the design. The directly-implemented design would be fine, given that no variability at the lower-level were introduced. But if sub-component variability emerges, the designer (or the implementer) may have to consider refactoring hard-coded fragments into smaller components.

We also find that this refactoring decision may not be made immediately on the arrival of the new requirements, but when more and more sub-component variability emerges or when the chief architect decides to merge the variations into the PLA. This is also the reason why not all the hard-coded variable features are refactored in real products. The refactoring is postponed as far as the architect can tolerate.

For example, a component named ExamMain [2] is responsible for controlling the examination process, such as initiating the examination UI, receiving examination paper file, invoking the ExamPaper interpretation component, packing answers, and sending packed answer file to the server. In the original design, receiving examination paper file happens *before* initiating examination UI. The sequence of the examination process is hard-coded. This sequence assumes that the examination paper is determined *before* an examinee logs in. But in a new marketing requirement (Supporting Multiple Examination Subjects in the Same Time Period), the examination paper cannot be determined until an examinee logs in, so that receiving examination paper file should happen *after* UI initialization. The first solution is simple: find the hard-coded two blocks for receiving file and initializing UI and then switch the two blocks in source code. It is OK for a quick response for the new requirement but really terrible for future maintenance. Therefore, an ExamProcess component is created to control the variability of the sequence of initiating UI and receiving exam papers when more variability requirements arrive and affect the related components. Note that the change of the sequence is not an upgrade, but a variation, because either sequence has some advantages. For the old sequence, examinees experience faster login, while for the new sequence, a mandatory function is implemented.

This fact shows that, in real development with limited resources, a higher-level component design may be directly implemented. Sub-component connections may be hard-coded. The reason is that architects and developers tend to spend minimal effort in software design, regardless of what may happen in "the future". This strategy is typical in SMEs, as in Wingsoft.

Meanwhile, variability, a key ingredient in SPL development, brings unpredictable architectural variants in various granularities. Sub-component variations which are caused by market requirements changes raise internal quality requirements that hard-coded "large" components should be broken into smaller components. If the variations are not considered in previous versions, the architectural design will have to be refined at a proper timing.

3) Adding new component to the architecture is rare, but every time it results in a new branch.

As discussed previously, new components may be introduced by splitting a large component into smaller ones. But we also observed that purely adding a brand new component into the architecture to provide new functionality does not happen very often. This means that the overall functionality of the product line is comparatively stable. This also implies that either the marketing requirements are comparatively stable or the product strategy guides the evolution of marketing requirements quite well during the SPL evolution.

Although it happens rarely, it seems that adding new components has always created a new product branch in our case study. There are only three brand new components that were added in the history of Exam-PL development. (1) **Camera component**. In November 2004, a tentative requirement stated: "Is it possible that the examination system takes a photo of the examinee?" Although the requirement is not mandatory, the project leader saw a series of potential requirements such as anti-cheating, video surveying, etc. Then the architect changed the architecture to accommodate the requirements by adding a new Camera module in the design. After that, two branches (variants) were created on that module. Although the TakePhoto feature was not formally delivered to the customers, the architectural changes persist. Finally, a recent release of EXT exams bound this feature. The latest upgrading in one branch is in May 2012. (2) **Switcher component** that supports switching from a full-screen examination UI and a corner-panel examination UI. In July 2011, a new requirement emerges, stating an examination with integrated functionality in Versions 23 and 24. The architect was not surprised at the new requirement because he had foreseen the potential requirement in March 2011 when a branch of the Examination product was split into two new products (i.e. Version 23 and Version 24). This requirement was only a trigger that convinced him to make the decision to add the new Switcher component. (3) **Multicast component** that enables audio multicast among examination clients. It is a variant component [3] of Audio Transfer in Figure 1. In November 2004, a customer proposed a new requirement that examinees may discuss on a topic via audio conversation on the network. As an experiment, the architect chose Multicast technology to implement online chatting. The design had been evolving for more than two years and stable for another two years.

To summarize, the reasons for adding new components are typically: Experimental implementation; Merging member products; and Creation of a new product series.

4) Urgent, tentative, and/or non-strategic requirements lead to temporary architectural clone.

In the evolution history of the SPL, we observed that the most appearing architectural evolution types are linear evolution and clone. Usually, clone creates a new branch of the product, while linear evolution upgrades the product within the same evolution branch.

[2] The name is a little bit different from that in Figure 1 only because this is the real code-based naming while Figure 1 is a sketch map for understandability.

[3] There are two variants of Audio Transfer. One is Multicast; the other is Broadcast. To save space, they are not depicted in Figure 1.

We also noticed that requirement change requests that repeatedly come from multiple customers have a better chance to be merged into the PLA. This phenomenon is noted for two reasons. One is that multiple occurrences of similar requirements change requests imply commonality in the market and merging it into the PLA conforms to the product strategy. The other is that the more architectural clones exists the more mutations there would be in the product line. There would be more and more propagations between any two product branches that increase the commonality among all member products.

Although making a *merge* decision is difficult (because it is very expensive in short-term evaluation), there must be certain points in time that merge should happen. This decision can be potentially supported by keeping track of and analyzing the arrival of both requirement change requests and architecture evolution instances.

B. How are market requirements affected by architecture design?

1) Most market requirements from current customers are agreed and implemented by introducing variations.

Although the project team has established an architecture and followed corresponding constraints under a long-term product strategy, current customer needs are fulfilled to the most extent by introducing variation points and variants to the architecture. This is a typical benefit that the team has got from adopting the SPL approach.

We have to admit that fulfilling customer needs is essential in modern software development, especially for SMEs. During nearly ten years' survival in the market, the Wingsoft Company has beaten several competitors by providing more flexibility and quick responses to requirement changes. The project team has also lost some customers because of not responding to gigantic requirement change requests that may have caused severe inconsistency in the architecture and would incur heavy development effort.

With the guidance of the product strategy requirement, the architect is ready for introducing variations to the architecture. Therefore, in most cases when new requirements came in, the architect either made local changes to the architecture (such as modifying a component's implementation, or breaking a large component into smaller ones, more flexible ones, etc.) or create an architecture clone for larger scale changes.

There are also cases that, when new requirement changes emerge, making an architectural change impact evaluation is difficult, so that the architect is not sure whether to agree to the requirements. One solution is to create an experimental version by architecture clone and do further experiments to determine whether the requirements are feasible. In most cases experimental results are positive so that the requirements are finally agreed and merged in the product line.

2) The architect did deny or postpone some customer requirement change requests, but usually would provide an alternative solution.

It is important for the architect to negotiate requirements with the customer (maybe directly, or indirectly through the project leader) and try to persuade the customer to accommodate current architecture design if the new requirement changes are customer-specific or too large scaled.

The architect is the one that knows the most about the architecture, so she is the one that should be responsible for the consistency of the PLA. She may be facing the customer directly or convincing the product leader that the requirements are problematic with the architecture and not worthy fulfilled.

It is a common trick to tell the customer "we will provide full support to your requirements in the next releases, but here is an alternative solution…". It usually works and the customer is not unhappy with the alternative solution. But our case study also shows that the requirements (denied directly or postponed temporarily) are highly possible to be raised in the future. Therefore, it is necessary to keep track of denied and postponed requirement change requests, and probably the architect should get somehow prepared for architectural changes.

3) Architectural variability help architects and product leaders to stand a better position in requirement elicitation.

There are two types of requirement elicitation in our case study: customer-oriented and future-market-oriented. For a customer, the exact needs are vague and hard to be expressed at the outset. This provides a space for architects and product leaders to build a concrete mental projection of the system by showing existing product instances and current architecture.

For future marketing, it is useful to collect various customer needs and examine whether combining different needs and variant components are meaningful. It will help product leader to build a dominant role in future requirement elicitation and also save effort negotiating requirements with future customers.

C. How can the analysis of requirement and architecture evolution history help future requirement elicitation and architectural design decisions?

There is a great deal of architectural knowledge and domain knowledge that are scattered in the evolution history of an SPL that are potentially very useful in decision making.

When a new requirement change request emerges, the architect or product leader need to decide: 1) if it is worthy to be fulfilled; 2) if yes, whether it can be fulfilled in existing solutions; if not, whether one can find a similar solution that can be used in negotiation with the customer; and 3) if the architecture needs to change, how can the effort be minimized.

Categorizing requirement change requests can support an architect to make an architectural design decision, as is

shown in Figure III. But it still remains a research question that the architect may need further support to decide whether it is appropriate timing to initiate an architectural evolution, what the evolution action should be, and to what extent the evolution action should be taken.

V. Conclusion

In this paper, we presented a case study on the evolution history of an industrial software product line – the Exam-PL, a subset of WES-PL developed by Wingsoft Company in China. We proposed a classification of requirement change types from the viewpoint of architectural impact in SPL development practice. Furthermore, we analyze the potential relationships between requirement change types and architecture evolution types. We find that in SPL development, requirement changes happen regularly and an architect may need more careful considerations before making a decision to respond to it. Meanwhile, given an SPL and several architectures of member products, an architect or product leader may have more bargaining chips to persuade a customer to follow a given architectural design, so that the PLA can be minimally changed.

This paper also presents the beginning of a research program towards understanding requirement evolution in SPL development. A comprehensive understanding of the requirement evolution may support architecture evolution decision making approaches in industrial SPL development and help architects keep a balance between market requirements and product strategy requirements. Deeper investigations to the modules and their evolution history are needed for future quantitative analysis. A solid methodological approach is also needed for understanding requirements categories and for assisting architecture evolution decisions, especially in software product line development practices.

Acknowledgment

This work is supported by National Natural Science Foundation of China (NSFC) under grant no. 60903013. The authors would also thank Mr. Xinrong Tang and Ms. Lizhen Cao in the Wingsoft Company for their help in the case study.

References

[1] N. Nurmuliani, D. Zowghi, S. P. Williams, Using Card Sorting Technique to Classify Requirements Change, 12th International Requirements Engineering Conference (RE'04), IEEE, 2004, 240-248

[2] J. D. McGregor, Agile Software Product Lines, Deconstructed. Journal of Object Technology, 2008, 7(8), 7-19

[3] Y. Wu, X. Peng, W. Zhao, Architecture Evolution in Software Product Line: An Industrial Case Study, the 12th International Conference on Software Reuse (ICSR 2011), LNCS 6727, Springer, 2011, 135-150

[4] A. Tang, P. Liang, V. Clerc, and H. van Vliet, Supporting Coevolving Architectural Requirements and Design through Traceability and Reasoning, in Relating Software Requiremens and Software Architecture, P. Avgeriou, P. Lago, J. Grundy, and I. Mistrik, Eds., 2011.

[5] A. Etien, C. Salinesi, Managing Requirements in a Co-evolution Context, 13th IEEE International Conference on Requirements Engineering (RE'05), IEEE, 125 – 134

[6] R. Stoiber, S. Fricker, M. Jehle, M. Glinz, Feature Unweaving: Refactoring Software Requirements Specifications into Software Product Lines, 18th International Requirement Engineering Conference, 2010 (RE2010), IEEE, 403-404

[7] A. Maccari, Experiences in assessing product family software architecture for evolution. the 24th International Conference on Software Engineering. ICSE 02. ACM. 585-592

[8] B. Nuseibeh, Weaving Together Requirements and Architectures, Computer, March 2001, 115-117

[9] N. Nurmuliani, D. Zowghi, S. Fowell, Analysis of Requirements Volatility during Software Development Life Cycle, proceedings of the Australian Software Engineering Conference (ASWEC'04), April 13-16, Melbourne, Australia, 2004, IEEE, 28-37

[10] N. Heumesser, F. Houdek, Towards Systematic Recycling of Systems Requirements, the 25th International Conference on Software Engineering, ICSE 03, IEEE Computer Society, 512-519

[11] M. Fischer, J. Oberleitner, J. Ratzinger, H. Gall, Mining evolution data of a product family, 2005 International Workshop on Mining software repositories, MSR 2005, ACM, 1-5

[12] M. Svahnberg,J. Bosch, Evolution in Software Product Lines: Two cases. Journal of Software Maintenance.1999. 11(6), 391-422