Architecture Evolution in Software Product Line: An Industrial Case Study

Yijian Wu, Xin Peng, Wenyun Zhao

School of Computer Science, Fudan University, Shanghai 201203, China
{wuyijian, pengxin, wyzhao}@fudan.edu.cn

Abstract. A software product line (SPL) usually involves a shared set of core assets and a series of application products. To ensure consistency, the evolution of the core assets and all the application products should be coordinated and synchronized under a unified evolution process. Therefore, SPL evolution often involves cross-product propagation and synchronization besides application derivation based on core assets, presenting quite different characteristic from the evolution of individual software products. As software architectures, including the product line architecture (PLA) and application architectures, play a central role in SPL engineering and evolution, architecture-based evolution analysis is a natural way for analyzing and managing SPL evolution. In this paper, we explore common practices of architecture evolution and the rationale behind in industrial SPL development. To this end, we conduct a case study with Wingsoft examination system product line (WES-PL), an industrial product line with an evolution history of eight years and more than 10 application products. In the case study, we reviewed the evolution history of WES-PL architecture and analyzed several typical evolution cases. Based on the historical analysis, we identify some special problems in industrial SPL practice from the aspect of architecture evolution and summarize some useful experiences about SPL evolution decisions to complement classical SPL methodology. On the other hand, we also propose some possible improvements for the evolution management in WES-PL.

1 Introduction

Reuse of unmodified components has not produced the promised rewards because the reusable components are seldom a precise fit to the reuse needs. One method of mitigating this shortcoming is to narrow the field of applicability to a software product line (SPL). An SPL is a group of products that share a common, managed set of features and that are developed from a common set of core assets in a prescribed way [1]. In the past decade, SPL is proven to be an efficient approach for both architecture- and component-level reuse.

Similar to individual software products, an SPL evolves continuously with requirement changes, scope extension and design refactoring. SPL evolution is usually more complex than the evolution of individual software products because both the core assets and a series of application products are involved. Successful SPL engineering requires management and coordination of the development of core assets

and application products to meet the organization's overall business goals [1]. To ensure the consistency and unity of the SPL, the evolution of the core assets and all the application products should be coordinated and synchronized under a unified evolution process. Therefore, SPL evolution often presents quite different characteristics from the evolution of individual software products.

The product line architecture (PLA) in an SPL specifies the common structure of all member products and centers in the development and evolution of both core assets and application products [2]. Therefore, an architecture-based approach is a natural way for analyzing and managing SPL evolution.

In order to explore SPL evolution practices, we conduct a case study on architecture evolution in the Wingsoft examination system product line (WES-PL), an industrial product line with an evolution history of eight years and more than 10 application products. In the case study, we reviewed the evolution history of WES-PL and revealed that real SPL evolution is usually much more complex than ideal due to practical difficulties. For example, ideal SPL practices suggest that all application products be derived from the PLA to keep consistency, but we find that some application products occasionally and purposefully deviate from the core assets for some reasons (such as market uncertainty). We categorize several common evolution types in SPL development and identify some special real problems in SPL evolution. We also gather useful experiences for evolution decisions from our historical study.

The rest of the paper is organized as follows. Section 2 introduces the business background of the WES project and shows an overview of the evolution history. Section 3 presents several evolution types discovered in the development history and discusses typical evolution scenarios in real development. Section 4 describes some high level findings and experiences in our study and proposes possible improvements to WES-PL development. Section 5 discusses related work in both industry and academia. Section 6 concludes our work.

2 Background

2.1 Overview of WES product family

WES is an industrial product family developed by Fudan Wingsoft Co. Ltd., a software company with about 50 employees in Shanghai, China. The WES project was started in late 2002 as a product development project for Shanghai Municipal Education and Examination Authority (SMEEA). It was first developed as a computer-aided oral examination software product [3]. In the following years, it gradually evolved into a complete WES product family, covering the whole business process of computer-aided educational examination as shown in Figure 1.

Each process in Figure 1 is supported by a product category (a set of WES products) for various kinds of examinations. These examinations include high-end ones, such as College Entrance Examination of the State, and low-end ones, such as final examinations in high schools. The examinations also have various educational purposes, such as oral exams, listening exams, debating exams, so on and so forth.

In the past eight years, the WES team has developed more than 16 application products to support different examinations. These products are listed in Table 1 with corresponding product category and a brief description. Each product has about 50~90K lines of Delphi code. Some products share more than 80% source code.

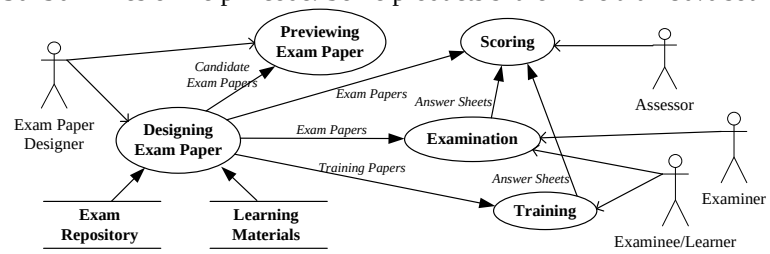

Fig. 1. Data flow chart of the computer-aided examination system

Table 1. A list of WES products developed

Product Category	No	Product	Description
Exam Paper Designing (EPD)	P0	EPDS: Exam Paper Design System	Various examinations are supported in several versions. The latest version covers all question types. This product provides complicated algorithms for making exam papers.
Examination	P1	SOLO	Each examinee orally answers questions played by the computer.
	P2	DISCUSS	Examinees are grouped in at most four and each group may discuss orally on a given topic via network.
	P3	DEBATE	Same as DISCUSS, except that the group is of two examinees.
	P4	LISTEN-SOLO:	Supports both listening comprehension (multiple choice questions) and oral examinations.
	P5	ONLINE	Supports internet-based listening and written examinations.
Exam Paper Preview	P6	PREVIEW	Gives a live preview of SOLO exam papers.
	P7	PREVIEW-LO	Same as PREVIEW but supports LISTEN-SOLO exam papers.
	P8	PREVIEW-ON	Same as PREVIEW but supports ONLINE exam papers.
Training	P9	TRAIN@home	Supports SOLO oral training. Training materials are delivered by compact disks (CDs). Learners typically use the product at home.
	P10	TRAIN@home2	Same as TRAIN@home but supports LISTEN-SOLO training.
	P11	TRAIN-ECLASS	Supports SOLO exams. Training materials are stored on a server. Typically used in classrooms with tutors' guidance.
Scoring	P12	ORALSCORE	Scoring for SOLO exams. Assessors listen to audio records of examinees and give a score anonymously.
	P13	DISCORE	Scoring for DISUSS and DEBATE exams. Preprocess is required to merge grouped examinees' audio into one stereo audio file. Assessors give scores to each examinee anonymously.
	P14	EClassSCORE	Scoring for TRAIN-ECLASS. Assessors listen to and score audio records of examinees, while name of each examinee is shown on screen. (i.e. not anonymously). Used in in-class guided learning.
	P15	GRAPHSCORE	Scoring for ONLINE exams. A graphical presentation of answers is shown to assessors.

The products are developed basically according to 1) supports of question types; 2) product category (e.g. examination, training, etc); and 3) how educational process will be brought out (such as self-learning or guided-learning). Among these products, EPD products shows most distinguishing features, since designing exam paper is quite different from showing them to the designers, examinees or assessors. Thus, the EPD

category is not considered in the rest of our study. Also, Scoring product shows different features; thus not considered either. We will discuss the criteria for selecting products in our case study in Subsection 2.3. Before that, we first take a quick look at the evolution history of these products.

2.2 A brief evolution history of WES products

The WES products listed in Table 1 were developed in different historical periods as shown in Figure 2 (excluding EPD products because they actually form up an individual product family). The time dimension spans from 2002 to 2010, and the product dimension covers four product categories. The evolution history of each product is denoted by a bar, whose left side showing the beginning of the product and right side fading out indicating the end of evolution. From Figure 2, it can be seen that some products like P2, P4, P7, P11 and P14 are no longer maintained (but may still in use with some customers).

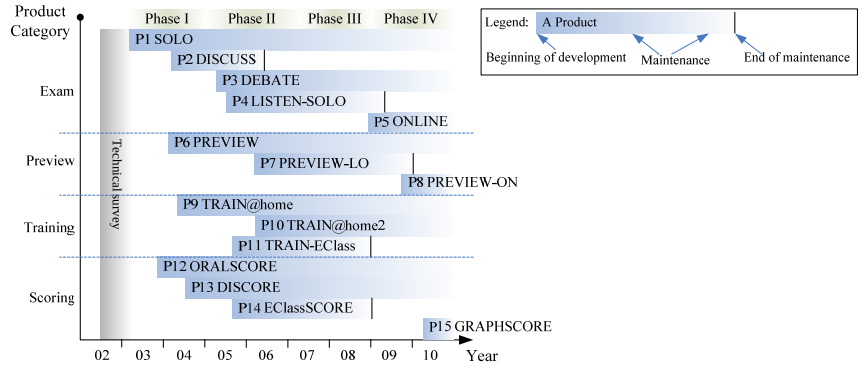

Fig. 2. Development history of WES-PL (Four phases)

The evolution history presented in Figure 2 can be distinctly divided into the following four phases.

Phase I: Startup (2003-2004)

Basic products of four product categories, i.e. P1, P6, P9, and P12, were developed, laying a foundation for future improvements. Moreover, products for group-discussion-exams (P2 and P13) were initiated due to market anticipation.

Phase II: Expanding (2005-2007)

Basic products were maturing. Meanwhile, new product series for group-discussion, debating and listening comprehension were released. In this period, the market of WES products kept expanding, and accordingly the WES team was often required to make quick responses to newly emerging or changing requirements. At the same time, some features became obsolete after delivery to market (e.g. P2 was not maintained since mid-2006) and was replace by some other products (e.g. P3).

Phase III: Maintenance and Refactoring (2008)

The theme of this phase was regular maintenance and refactoring. As traditional computer-aided examination market had been nearly carved up, no new products were introduced in this phase. Therefore, the WES team had plenty of time to plan and

conduct design refactoring on both the domain and application levels, providing better services or user experiences in existing products.

Phase IV: New Era of Development (2009-2010)

The WES team received new market opportunities, which is a driving force for deeper refactoring and future development. New WES products combining the features of listening and writing examinations were planned and developed by reusing existing software assets. The PLA was actually refactored and the products are gradually synchronized with the PLA to achieve higher development efficiency.

2.3 Scope of our case study

Systematical architecture-based and comprehensive reuse-based product development is an essential characteristic of SPL engineering. Therefore, we confine our case study to Examination, Training and Preview products, because they share the essential common features of exam paper packaging, distribution, displaying and answering, and share a common PLA. EPD products and Scoring products, although mentioned together with the WES product family and sharing *some* commonalities with other products, are not included in the scope of our case study, since they do not share architecture-level commonality with other produces. Thus, eleven (11) products, i.e. P1~P11, are included in the scope of our case study. Among these products, Examination products (P1~P5) share almost the same architecture. The architecture of Preview products (P6~P8) is approximately a subset of corresponding Examination products (P1, P4 and P5). Training products P9 and P11 extend the architecture of P6 and are modified based on P1; P10 is built based on P7 and is modified based on P4.

In the following sections, WES-PL specifically denotes Examination, Training and Preview products.

3 Architecture Evolution in WES-PL

For better understanding of the architecture evolution history, we first give a brief introduction to the most recent PLA skeleton of WES-PL. Then we describe the architecture evolution types discovered in WES-PL evolution history. After that, we present an architecture evolution roadmap of WES-PL with detailed description of some typical evolution cases. Finally, we summarize architecture evolution in WES-PL with both qualitative and quantitative analysis.

3.1 The PLA skeleton

The most recent PLA skeleton is presented in Figure 3, showing commonality and variability in all concerned products. The basic structure includes six variable components (i.e. Network, Server, GUI, Authentication, and PaperReader) with several variants each and other optional and variable accessory components (such as audio and video components). The major architectural variations are described as the following: 1) Network protocol: Some products work on TCP/IP protocols; some

others work on HTTP protocol via Internet; even others do not need a network component (e.g. the Preview products). 2) Server component: Examination and Preview products need EPC while Training products need LCM. There are also several variant for EPC and LCM. 3) GUI suite: Different products adopt various GUI suites for different examinations; also, user interactions provided by UI suites are different (e.g. Examination products typically provide only limited control for end users, while Training or Previewing products provide more freedom for end user interactions). 4) A/V processors: The PL has adopted several different audio/video components; some products need voice multicast; video playback support is an optional for certain examinations. 5) Other variants are trivial, such as Database, Authentication and Exam Paper Reader.

The PLA has been evolved for eight years, and is very likely to be evolving in the future. The skeleton is a quick snapshot of current PLA. In the following discussion, we actually address the problem that how *this* version of PLA is achieved.

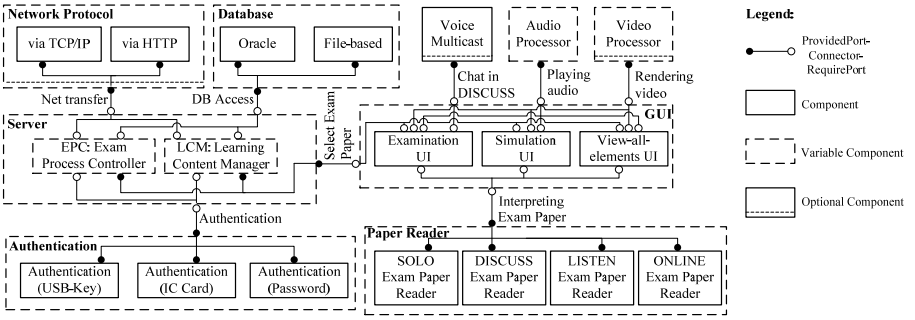

Fig. 3. The PLA skeleton of WES-PL

3.2 Typical architectural evolution types

In our previous work [2], we have indicated that, besides application derivation, there are continuous interactions between application architectures and the PLA during the evolution of the PL. We now further categorize these architecture evolutions into six types, namely *linear evolution*, *derivation*, *merge*, *synchronization*, *clone* and *propagation*.

- **Linear evolution:** Linear evolution is driven by intrinsic factors of an application or the PLA.
- **Derivation:** Derivation is to create the initial architecture of a new application from the PLA by variability customization and extension.
- **Merge:** Architecture merge is a kind of periodic feedback from application architecture to the PLA. When merging, architects collect and analyze architectural differences within all current application architectures and resolve possible architectural conflicts before changes are integrated to the PLA. Note that the PLA may evolve after a merge to prepare for a synchronization to all products (see Synchronization below). Before such a follow-up linear evolution, all application

architectures have to be frozen in order to prevent incompatibility or inconsistency. Such a "frozen" is also depicted in Figure 4.

- **Synchronization:** Synchronization is to reconcile application architectures with the PLA. Usually, architectural conflicts should have been resolved before synchronization.
- **Clone**: Architecture clone creates the initial architecture of a new application from an existing application architecture. Architecture clone allows application architectures to deviate from the PLA, although such deviations is ultimately to be resolved.
- **Propagation:** Architecture propagation is to propagate architectural modifications among application architectures. Modifications to an application architecture could be experimentally applied in sibling application architectures before merged into the PLA.

A typical architecture evolution trace in a PL with these types can be exemplified in Figure 4, followed by a short summary for each evolution type in Table 2.

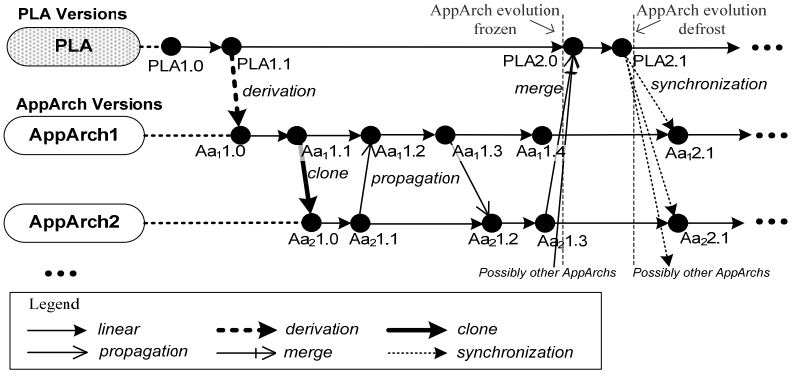

Fig. 4. Illustration of architecture evolution types

Table 2. A summary of evolution types

Evolution type	Modification		Comment
	From	To	
Linear evolution	AppArch	**next version** of AppArch	Usually involves proactive reuse considerations to better feed future application development
	PLA	**next version** of PLA	Usually reflects a quick response to changes in requirements or environment
Derivation	PLA	**new** AppArch	A traditional way to create a new application from the PLA
Merge	AppArchs	**next version** of PLA	Modifications on AppArchs are collected in the PLA
Synchronization	PLA	**all** existing AppArchs	New design features are spread from the PLA to all applications
Clone	AppArch	**new** AppArch	An informal but useful way to create a new application from an existing application
Propagation	AppArch	**some** other existing AppArchs	Architectural changes to an application can be experimentally applied to another application

With these types in mind, we further analyze the history of WES-PL architecture evolution and find out how these evolution types actually take effect in real development.

3.3 A roadmap of WES-PL architecture evolution

In this subsection, we show a more detailed evolution history than that in Section 2.2. We do not investigate every revision in the evolution history, but focus on primary milestones of each product. Figure 5 shows such a history, with the eleven products in the scope, indicating the *overall* complexity of the evolution trace. The vertical axis represents concerned products, while the horizontal axis indicates time.

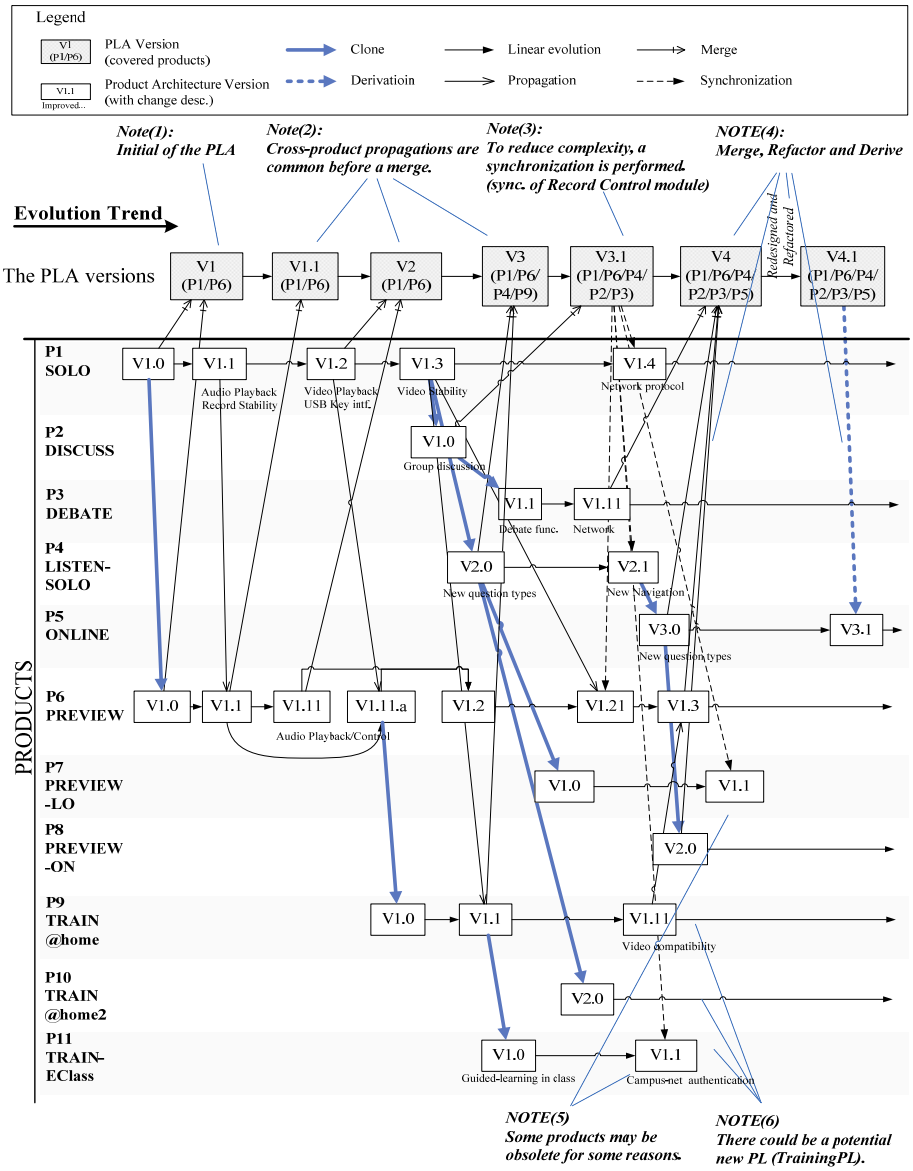

Fig. 5. WES-PL architecture evolution history (since early 2003 till early 2010)

Several special cases depicted in Figure 5 as *NOTE(n)* are worthy of mentioning. The rationales behind will be discussed in subsection 3.4.

- *NOTE(1): Initial of the PLA.* The PLA is created from the first merge. In our case, we find that the PLA is not designed purely in domain engineering processes, but merged from P1 and P6.
- *NOTE(2): Propagations before merges.* Architectural changes are usually propagated to at least one other product before being merged into the PLA. PLA versions 1.1, 2 and 3 take changes that have been made to at least two products.
- *NOTE(3): Synchronization for lower complexity.* At this point of time, developers maintain several versions of sound recording control in at least five products. In order to reduce the complexity, all related products are synchronized on the Sound Recording Control subsystem after the new mechanism is proved to be usable and effective in the previous versions of P2 and P9.
- *NOTE(4): Merge, Refactor and Derive.* Only a carefully designed architecture is suitable for derivation. A derivation happens on PLA Version 4.1, rather than on PLA Version 4 which is immediately after a merge. A careful redesign is made to PLA v4 to achieve better maintainability.
- *NOTE(5): Obsolete products.* P7 is obsolete at the point of time when no more customers are interested in its functionality. Most customers are interested either in more stable, old products, or in the next generation of a new product (P8). Also note that P8 is not cloned from P7, which is in the same product category, but from P5, the examination correspondence with more similarity. Also, P11 is temporarily obsolete for different reasons that potential users have not been inspired and that current products (P9 and P10) have already fulfilled most of current requirements.
- *NOTE(6): New product lines may be identified.* P9~P11 are getting more and more different than other products. While they are getting "too different", the architect is deciding to create a new PL for these products.

These practices give a quick glance at the industrial product line development reality. More detailed analysis will be shown in the following subsection.

3.4 Summary of architecture evolution in WES-PL

A quantitative and qualitative summary of the evolution practices is listed in Table 3.

Table 3. An analysis for evolution types

Evolution type	Count	Comment
Linear evolution	many	Linear evolution is trivial in the whole development process. We only consider major builds of products and versions of the PLA.
Clone	10	New products are typically cloned from similar products, rather than derived from PLA.
Synchronization	>5	Existing products may take part of the modifications from the PLA. Future products are not explicitly affected.
Derivation	1	In our case, "derivation" happens only when the new online examination system is developed.
Merge	>12	Architectural modifications are merged into the reference architecture. The reference architecture then covers several products.
Propagation	>5	There are at least 20 modifications propagated among products, but most of them are between small revisions. If products share similar modules, then a modification to that module will propagate to other products containing the same modules.

As we review this history, we can find the answers to the following questions: First, how a new product is created; second, how a new product line is created; third, how the PLA evolves; and finally, what tends to be typical evolution decisions in an industrial SPL development in the real world.

- Clone vs. Derivation

In real development, both derivation and architecture clone are ways of creating a new product. Derivation from the PLA is regarded as a good and standard practice in SPL development. But in our case, developers typically do not derive new products from the PLA. Rather, they tend to create products by cloning the architecture (usually also with source code) of an existing product. The reason is that cloning is usually cheaper in early development, especially when the team of developers is stable. Moreover, a complete domain engineering process is usually not available due to limited resources in the company. Thus, blindly deriving a product from the PLA is only too risky for most developers. Therefore, the reality is that most new products are built from older products, rather than derived from the PLA.

- Initiating a product line with actual products

The beginning of all these eleven products is not a design of the PLA, but actual working products P1/P6. In real development in a medium-small company, there are not enough time and resources to trade a working product for a PLA. It is only acceptable that, when there *are* several products working, the PLA can be tentatively built and incrementally evolve. The PLA is not necessarily the source of all products in the SPL, but provides a broader vision for all future products.

Meanwhile, new PLs may be created by "splitting" current PL when the application architecture shows too many differences. As is noted in Figure 5, a Training-PL may be created by considering all Training products in the future.

- The evolution of the PLA

There are two cases that drive PLA evolution: 1) merge and 2) redesign. We find that architectural changes in the PLA are primarily driven by merge from various products. In our case, the architect did not frequently directly modify/redesign the PLA actively. He believed that a modification to the architecture was risky; therefore, unless there was a significant reason (say, for improving performance or for a clear and splendid future), the architect was not willing to modify the PLA proactively[6].

If such a significant reason is present, architectural refactoring happens periodically and inevitably, which triggers a pure linear evolution of the PLA.

In practice, a relaxed approach is adopted: modifications are experimentally made to a single product. Such modifications are cheap and safe, because they are restricted within one single product. If the modifications are proven to be not useful or not acceptable to the market, the features are simply abandoned, and the PLA remains untouched.

- The PMS pattern: accepting new features in SPL

We find in our case that a typical path to add a new feature to all products follows the Propagating-Merging-Synchronizing (PMS) pattern, meaning that a feature should not be integrated in the SPL unless it is propagated to more than one product.

First, a linear evolution is carried out for the specified product (Product A). Then the modification is propagated to another product (Product B) sharing the same components relative to the modifications. If Product B shows the modification reasonable, then the modification is a candidate to be collected in the PLA. After the

architect identifies the new feature as a common one, the modification is then merged in the PLA. Before the modification is synchronized to all other products, the architect should further evaluate the quality of the new architecture. Typically, a refactoring will be carried out, leading to a linear evolution of PLA. After that, the new features can be finally synchronized to all other products in the SPL.

4 Evaluation and Discussion

Before evaluation and discussion, we have to mention that, although the architecture evolution trace observed in our case study with the six evolution types faithfully reflects the actual development history, we still find some limitations in our study. One is that our case is with a medium-small software company developing products of its own. Therefore, our case may not be applicable to large, international software companies or outsourcing companies. Another is that our study is based on only one PL developed continuously by one team. Some complicated architectural evolution phenomena (such as those with collaboration issues) may not be observed.

The threats do exist and may provide some new perspectives for our future study. However, we can still elicit common practices valuable within the scope of our case study. In the following discussions, we highlight some special problems in industrial SPL practice from the aspect of architecture evolution and discover some useful experiences about SPL evolution decision to complement classic SPL methodology. On the other hand, we also realize that the current development practice in WES-PL is far from perfect, so we would like to propose some possible improvements for evolution management in WES-PL.

4.1 Proactive Evolution vs. Reactive Evolution

McGregor [7] describes two approaches in SPL engineering: the proactive approach, which leads to a risk of developing possible useless assets, and the reactive approach, which implies more effort in reactively prepare assets for later reuse. Our case confirms this conclusion. Moreover, we also find that, regardless future conformance with PLA, reactive approach may achieve short time-to-market because short-sighted modification may be more light-weighted and thus captures short-termed opportunities (such as in a bidding).

Proactive reuse-based strategy is the foundation of SPL development, promising benefits like improved quality and reduction of development costs and time to market. However, reactive strategy can significantly reduce the upfront investment, requiring closer coordination within the SPL project [1].

In fact, after an SPL is established, both proactive and reactive evolutions exist. In proactive evolution, feedbacks from application engineering to domain engineering are also allowed, but evolution decisions must be initiated on the PLA before applied to application architectures. In Figure 5, proactive architecture evolution only involves architecture derivation and synchronization besides PLA linear evolution. On the contrary, reactive architecture evolution allows application architectures to

temporarily deviate from the PLA by clone or self evolution and then synchronize with the PLA by consequent merging and synchronization.

Apparently, proactive evolution is consistent with the SPL principle of proactive reuse, thus should benefit the SPL promises on quality, cost and time to market. However, we find reactive evolution has its rationality in SPL practice, especially in small or middle-sized SPL projects facing intense market competition. The SPL promise of reducing time to market by proactive reuse is on the premise of accurately predicting emerging and changing product requirements. However, in a competitive and rapidly changing market like computer-aided online examinations, such long-term predictions are not realistic. For example, when preparing for a bid, urgent new features may come from competing products and/or potential customers (usually the bid inviters). In such a situation, the SPL team is always under the pressure of rapid responses to the new features even if the new features are out of the scope of the PL. Thus, application engineers usually choose to clone from a most similar exiting product rather than start a proactive evolution process.

Besides response time, the risk of overdesign for uncertain features can also be reduced by reactive evolution. Due to immaturity of the market and uncertainty of requirements, newly planned features may finally be abandoned. Implementing such a feature in an experimental product would be much cheaper than incorporating it in the PLA as a variation point. If the uncertain new feature is validated in one or more products before incorporated into the PLA, unnecessary time and efforts would be saved.

Reactive evolution may make application architectures deviate from the PLA. In order to reconcile them, architecture merge and synchronization should be conducted, usually when the whole SPL is relatively stable. This means that new features have been fully validated by the market, and that the whole SPL team can focus more on refactoring the PLA and coordinating all application architectures with the PLA.

4.2 Business Strategy and Technical Decisions

SPL methodology is market-oriented, so technical decisions should undoubtedly support and serve the business strategy. The technical decision of proactive or reactive evolution discussed above actually reflects this principle. The experiences from WES-PL confirmed that the technical choice of whether adopting a proactive approach or a reactive approach largely depends on the market position of the SPL organization. If it has the leading position in the target market, the organization usually takes the initiative in SPL evolution and prefers a proactive strategy. On the contrary, if it is in the expanding stage struggling to earn a place in the market, the team often passively absorbs emerging ideas from competitors and customers. Therefore, the reactive evolution strategy is better for rapid responses to emerging features and reducing the risk of misestimating the evolution trends.

On the other hand, technical decisions can adversely affect the business strategy of the organization by providing feedbacks. The technical decisions have great influence on the cost aspect of business considerations, and the influences are usually hard to be evaluated at the business level. For example, the scoring products were initially included in the WES-PL from the business perspective, since they share some

components in common with examination products. However, according to the feedback from reactive architecture merging (not shown in our case study), too many differences are found and too much complexity is included. Thus, scoring products were excluded from the WES-PL finally. A similar demerge chance can be found in the Training product category, which is mentioned as a "split" in subsection 3.4.

4.3 Possible Improvements

Proactive evolution and reactive evolution essentially reflect the tradeoff between unity and agility in SPL development. In WES-PL, reactive evolution decisions like architecture clone and propagation are adopted for agility. Architecture merge and synchronization should be continually enforced to keep balance between unity and agility.

However, we find the current development practice in WES-PL emphasizes agility over unity too much. Related problems include 1) loose control on reactive architecture evolution, 2) long synchronization cycle and 3) incomplete synchronization. Intense market competition and tight time arrangement make the SPL team overuse reactive evolution approaches, which partly causes these problems.

Loose control and long synchronization cycle are resolvable by adjusting team organization and development process. But such adjustments should be based on a stable profit of existing products to the company. The problem of incomplete synchronization can be resolved by PLA refactoring. As inconsistent designs exist across the application architectures, PLA refactoring involves adaptations to the variability design to accommodate the inconsistencies among application architectures and deviations away from the PLA. We find the PLA refactoring practices are not well established in WES-PL.

All these problems reflect the shortcomings of reactive evolution in WES-PL, i.e. the basic role of the PLA in reuse-based SPL engineering is crippled. If no effective measures taken, the product line may degenerate to a series of individual product projects with manual copy/paste-based code reuse. Based on the above analysis, we suggest possible improvements to the current WES-PL practice. From the aspect of management, we suggest strict control over reactive evolution. The synchronization cycle should be shortened and formulated in the organization or within the team. From the technical aspect, we suggest more effort on PLA-level refactoring to better reconcile application architectures and the PLA.

5 Related Work

There are several industrial case studies on SPLE, focusing on adopting SPL methodology in software development [8,10,13,14,15]. But there is not yet a widely adopted or general approach for industry to easily and efficiently transit from traditional single product development to software mass customization [9]. There are also challenges in contemporary industrial SPL development in managing continuous changing and emerging variabilities [10]. In a newly created product line, changes to the reused core assets and their customized instances also need to be efficiently

propagated and managed [11]. Our case study actually shows such attempts that a medium-small company focusing on a particular market tried to adapt the SPL methodology in the past eight years to maximize the throughput of customized products.

Svahnberg and Bosch [12] conducted a detailed evolution analysis on two software product lines with a history of nearly 10 years since 1990. Among several generations and releases of products, they identified several categorizations of SPL evolutions on requirements, architectures and components, and provided some guidelines for software product line evolution. We have adopted a similar methodology with their work. Our case study, however, tries to find some special practices other than standard PL development. We explore deep into one particular product line, and focus specifically on the architecture evolution. We believe our findings in our case present more recent SPL engineering in medium-small companies, including both standard and agile practices.

There are also researches and case studies on interactions, integrations and feedbacks between single products and common assets in an SPL, including military development with US Navy [16] and commercial organizations [1,17,18,19]. Our case study further discusses what exactly these interactions and feedbacks are, and what other interactions exists between the core assets and instantiated applications.

More research and industrial groups reported their SPL engineering and reengineering practices, showing adoption SPL methodology in traditional development. These research and practices include adopting product line development without the use of a PLA [20], reusing legacy components in a product line [21], and migrate legacy systems into product lines [22,23]. There are also researches on incrementally introducing features in product line development with proactive scheduling and road mapping [24].

In these cases and researches, we find side effects when adopting non-standard SPLE process, but such practices also bring some effectiveness to industrial development.

Despite of all these non-standard SPLE practices, researchers pointed out that SPLE in commercial organizations is ultimately evolving towards high maturity, both technically and organizationally [25,26]. Our WES project is actually at a relatively low maturity level and evolves towards higher maturity, as the PLA gradually integrates all features and variability from other existing or potential products. The evolution direction of an SPL can be anticipated by long-term forecasting [27], contributing to SPL design decisions in current development. This is why we believe the possible improvements proposed in Subsection 4.3 are reasonable.

The SPL development in Wingsoft also shows something in common with agile software product line method [7]. A successful integration of SPL development practices and agile software development practices is described by M.A. Babar [28].

In our previous work, Peng [2] has outlined an architecture-based evolution management method, which illustrates several architecture evolution cases. In another product line with Wingsoft, we identified common implementation mechanisms for product variations [4]. We also tried to manage interleaved interactions between the core assets (including PLA) and application architectures (including instantiated artifacts) [5].

6 Conclusion and Future Work

In this paper, we report an industrial case study on architecture evolution in SPL engineering. We identify a series of architecture evolution types and several typical evolution paths. Some evolution types, such as propagation and clone between products, usually causing architectural deviations, are quite common in the evolution history, showing a typical reactive style. We find that such reactive evolution plays an important role in small or middle-sized SPL projects facing intense competitions and market uncertainty. Although not conforming to classical SPL methodology, reactive evolution has rationality in SPL practice due to its rapid responsibility to emerging features and low risk of overdesigning. To keep the unity across all products, periodical merges and synchronizations are performed. In particular, architectural unifications are reasonably derived from specific variant products, rather than totally designed proactively. Based on the industrial case, we also propose some possible improvements for the evolution management in similar industrial product lines. Although our experiences may not apply to large, international PL development, we believe our case represents the reality of PL practices in medium-small companies.

Our case study highlights some real industrial SPL practices on the perspective of architecture evolution and provides useful experiences about SPL evolution management to complement classical SPL methodology. In our follow-up study, we plan to introduce knowledge-based method to model SPL design decisions and explore knowledge-based SPL evolution analysis from the perspective of design decisions.

Acknowledgments. The work presented is supported by National Natural Science Foundation of China (NSFC) under grants 60903013, 90818009 and 60703092. The authors would also thank Xiaofeng Qian and Shunxiong Ma in Wingsoft for their help in the case study.

References

1. Clements, P.C., Jones, L.G., Northrop, L.M., McGregor, J.D.: Project Management in a Software Product Line Organization, IEEE Software, 22(5), 54-62 (2005)
2. Peng, X., Shen, L., Zhao, W.: An Architecture-based Evolution Management Method for Software Product Line. In: SEKE 2009, 135-140. KSI Graduate School, IL (2009)
3. Wu, Y., Zhao, W., Peng, X., Xue, Y.: A Concept Model for Computer-based Spoken Language Tests. In: AICT-ICIW 2006, 19-24. IEEE Computer Society (2006)
4. Ye, P., Peng, X., Xue, Y., Jarzabek, S.: A Case Study of Variation Mechanism in an Industrial Product Line. In: ICSR 2009, 126-136. Springer (2009)
5. Shen, L., Peng, X., Zhu, J., Zhao, W.: Synchronized Architecture Evolution in Software Product Line using Bidirectional Transformation. In: COMPSAC 2010, 389-394 (2010)
6. McGregor, J.D.: The Evolution of Product Line Assets. Techinal report, CMU/SEI-2003-TR-005, ESC-TR-2003-005 (2003)
7. McGregor, J.D.: Agile Software Product Lines, Deconstructed. Journal of Object Technology. 7(8), 7-19 (2008)
8. Jiang, M., Zhang, J.: Maintaining Software Product Lines – an Industrial Practice. In: ICSM 2008. 444-447 (2008)

9. Krueger, C.W.: Easing the Transition to Software Mass Customization. In: the Fourth International Workshop on Software Product-Family Engineering. LNCS vol. 2290, 282-293. Springer, Heidelberg (2001)
10. Chen, L., Babar, M.A.: Variability Management in Software Product Lines: An Investigation of Contemporary Industrial Challenges. In: 14th International Conference on Software Product Lines. LNCS 6287. 166-180. Springer (2010)
11. Anastasopoulos, M.: Increasing Efficiency and Effectiveness of Software Product Line Evolution–An Infrastructure on Top of Configuration Management. In: Joint International and annual ERCIM Workshops on Principles of Software Evolution (IWPSE) and software evolution (Evol) workshops. 47-56. ACM (2009)
12. Svahnberg, M., Bosch, J.: Evolution in Software Product Lines: Two cases. Journal of Software Maintenance. 11(6), 391-422 (1999)
13. Bosch, J.: Product-line architectures in industry: a case study. In: ICSE 99. 544-554. ACM, New York (1999)
14. Maccari, A.: Experiences in assessing product family software architecture for evolution. In: ICSE 02. 585-592. ACM, New York (2002)
15. Axelsson, J.: Evolutionary Architecting of Embedded Automotive Product Lines: An Industrial Case Study. In: WICSA09. 101-110. IEEE, Cambridge (2009)
16. Brownsword, L., Clements, P.: A Case Study in Successful Product Line Development. Technical report. CMU/SEI, CMU/SEI-96-TR-016 (2006)
17. Riva, C., Rosso, C.D.: Experiences with Software Product Family Evolution, In: 6th International Workshop on Principles of Software Evolution. 161-169. IEEE (2003)
18. Takebe, Y., Fukaya, N., Chikahisa, M., Hanawa, T., Shirai, O.: Experiences with software product line engineering in product development oriented organization. In: SPLC 09. 275-283. CMU, Pittsburgh (2009)
19. Lee, H., Choi, H., Kang, K.C., Kim, D., Lee, Z.: Experience Report on Using a Domain Model-Based Extractive Approach to Software Product Line Asset Development. In: ICSR 2009, 137-149. Springer, Heidelberg (2009)
20. Staples, M., Hill, D.: Experiences Adopting Software Product Line Development without a Product Line Architecture. In: 11th Asia-Pacific Software Engineering Conference. 176-183. IEEE (2004)
21. Kolb, R., Muthig, D. Patzke, T., Yamauchi, K.: A Case Study in Refactoring a Legacy Component for Reuse in a Product Line. In: ICSM05. 369-378. IEEE Press (2005)
22. Breivold, H.P., Larsson, S., Land, R.: Migrating Industrial Systems towards Software Product Lines: Experiences and Observations through Case Studies. In: 34th Euromicro Conference Software Engineering and Advanced Applications. 232-239. IEEE (2008)
23. Hanssen, G.K.: Opening Up Software Product Line Engineering. In: the 2010 ICSE Workshop on Product Line Approaches in Software Engineering, 1-7. ACM (2010)
24. Savolainen, J., Kuusela, J.: Scheduling Product Line Features for Effective Roadmapping. In: the 15th Asia-Pacific Software Engineering Conference, 195-202. IEEE (2008)
25. Bosch, J.: Maturity and evolution in software product lines: Approaches, artefacts and organization. In: SPLC2002, LNCS 2379, 247-262. Springer (2002)
26. Ahmed, F., Capretz, L.F.: An organizational maturity model of software product line engineering. Software Quality Journal, 18(2), 195-225 (2010)
27. Chen, Y., Gannod, G.C., Collofello, J.S., Sarjoughian, H.S.: Using simulation to facilitate the study of software product line evolution. In: 7th International Workshop on Principles of Software Evolution, 103-112. IEEE (2004)
28. Babar, M.A., Ihme T., Pikkarainen M.: An Industrial Case of Exploiting Product Line Architectures in Agile Software Development. In: SPLC 2009, 171-179. CMU, Pittsburgh (2009)