An Organizational Approach for Industrialized Systems Integration

Matthias Minich, Bettina Harriehausen-Mühlbauer, and Christoph Wentzel

h_da – University of Applied Sciences Darmstadt, Germany
matthias.minich@web.de
b.harriehausen@fbi.h-da.de
c.wentzel@fbi.h-da.de

Abstract: Software development in systems integration projects is still reliant on craftsmanship of highly skilled workers. To make such projects more profitable, an industrialized production, characterized by high efficiency and quality, seems inevitable. While first milestones of software industrialization have recently been achieved, it is questionable if these can be applied to the field of systems integration as well. One of the most important concepts herein is specialization, represented by Software Product Lines. The present work analyses this concept against the particularities of systems integration and subsequently develops an alternative approach suitable for its implementation. The outcome is a three-layered organizational model that adapts and distributes the processes of Software Product Line Engineering and Product Development in accordance with the requirements of systems integration.

1 Introduction

Software Engineering offers several methodologies for industrialized software development, representing specialization, standardization, systematic reuse, and automation. The latter three are based on standardization to be most effective, which is represented by Software Product Lines (SPL). It seems to be very difficult or even impossible to create reuseable artefacts or automate development in an arbitrary context. A Software Product Line therefore spans a clearly delimited frame around a family of software products, sharing a common set of features and artefacts within a particular segment [CN07]. By concentrating on a limited scope, reusable production assets can be much more powerful than in a generic one. Unfortunately this does not apply to all fields of software engineering.

With reference to systems integration, the multiplicity of different technologies, caused by high heterogeneity, inflexible legacy systems and different data sources, seems to be a major drawback to the definition of distinguished product lines. In a product line covering Supply Chain Management systems, products may be highly integrated with third party shopfloor or finance systems, for instance. Including support for any potentially attached system undermines the advantages of a delimited context, while ex-cluding them will force development to occur outside the industrialized concepts. Another major drawback is the de-facto development of one-off products per customer.

Barely any solution operates in the same environment or is interconnected with the same type of systems. The initial setup cost for software product lines may therefore be contraindicative as the return of investment cannot be ensured.

2 Organization of Product Lines

Software Product Lines separate between product and product line development, represented in the following two scenarious:

A) Quality oriented development of standard software in a complex environment with a high degree of novelty. Considering the impact of a defect introduced into the underlying product line, a high level of quality outweighs the objective of lower cost. It is assumed that developing a new software product line is highly complex [Lin07] and comes with a high degree of novelty. As of a product line's fundamental and long lasting nature, a low dynamic is assumed.

B) Cost oriented development of customized software in an established environment with low dynamics and low complexity, which applies to product development. As most motives for software product lines are based on economic considerations [Lin07, CN07, PBL05], cost outweighs quality in this scenario. Furthermore, a certain level of quality is automatically ensured by utilizing artefacts of the respective product line. As products are developed within the boundaries of a well-known environment, novelty and dynamics can be considered as low. Likewise, complexity is reduced due to the reuse of common parts in a predefined platform and architecture.

Out of these scenarious, the present work developed a generic organizational structure for software product lines. It is thereby considering several works on organizational structures from an economic and software development point of view [Gro95, WD08, Töp85, Fre98, Lan04].

2.1 Software Product Line Engineering

Software product line engineering consists of several processes to define its scope and develop the infrastructure and core assets to be utilized in future products. Considering literature on software product line engineering [PBL05, CN07, Lin07, Bal08], the primary processes can be subsumed as follows:

- *Business Domain Analysis* identifies typical business processes, associated problems and solutions, and evaluates and prepares this knowledge for further processing. It also specifies a product's features within a software product line.
- *Domain Requirements Engineering* defines a product line‘s scope by identifying products and documenting their commonalities and variabilities. This scope may evolve over time [CN07].
- *Architecture Design & Development* transforms the scope into a technical architecture for the product line and its products. The architecture decomposes a

software system into common and variable functional parts, defines relationships and interfaces, and establishes rules for their implementation.

- *Core Asset Development* provides detailed design and implementation of reusable components based on the reference architecture [PBL05]. It includes executable code, variability mechanisms, common processes, development tools, and any other reusable assets.
- *Domain Testing* develops test cases and inspects all core assets and their interactions against the requirements and contexts defined by the product line architecture. This also includes validation of non-software core assets.
- *Software Integration* occurs during preintegration of several software components. They form blocks of functionality common to all products and contexts of a product line. It also ensures the interoperability of all reusable assets and provides the required integration mechanisms.

Implementing a delimited software product line is a singular undertaking for an enterprise. Although the primary processes remain the same throughout the development of other domains, work objects (architecture or core assets for instance) are unique and require different production steps for their completion. A decomposition based on work objects thus seems more appropriate than an activity based breakdown. It results in divisional units, responsible for all tasks required to create or modify a heterogeneous and potentially dynamic work object [Gro95]. This decomposition however, is limited by the interdependencies of the underlying work objects. It must be decided if interacting objects should be merged and represented in a single organizational unit, or if they can remain separate in favour of smaller and more efficient units. For software product line engineering, the present work suggests a structure as depicted in Figure 1. It assumes a low interdependency of business domain analysis due to its observing and strategic focus. It is therefore represented in its own organizational unit. Requirements Engineering and Architecture Design & Development are joined as of their intense interaction during problem definition and solution development within a new problem domain [Lan04]. Once requiremens and architecture are defined, core asset development may occur. It is expected to have only litte interdependencies as their overall structure and requirements have been defined in the previous two processes. They are implemented in their own unit and may be parallelized. Integration testing requires a certain degree of integration to be performed. It is therefore joined with the software integration process.

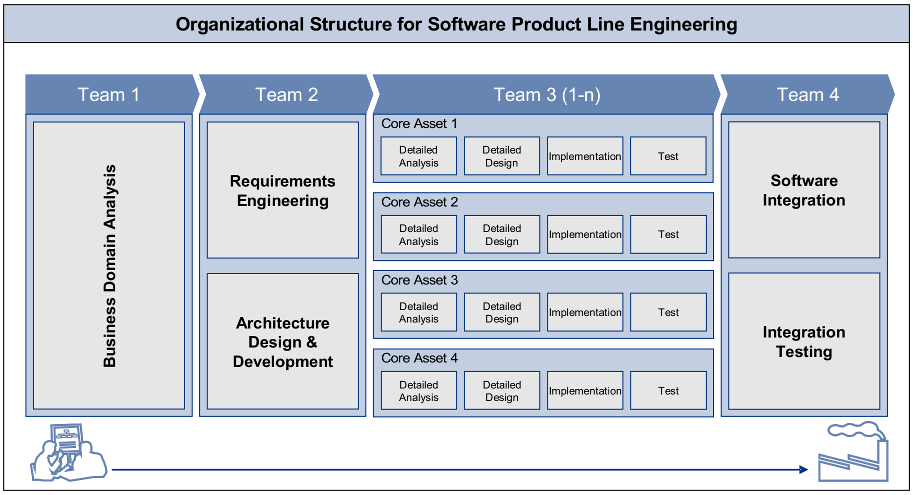

Figure 1: Organizational Structure for SPL Engineering

2.2 Product Development

In product development, the applications of the product line are built by reusing the core assets developed in software product line engineering. Considering literature [PBL05, CN07, Lin07, Bal08], the primary processes of product development can be subsumed as follows:

- *Product Requirements Engineering* analyses the variance of product requirements from the product line's core assets and decides whether to implement application specific assets or accept a functional trade-off.
- *Product Design* derives a particular product architecture from the overall product line architecture. Abstract variation points are instantiated and product specific requirements added. It defines how the product will be realised and may be compared to a detailed technical concept in single system development.
- *Product Realisation* assembles the application from core assets within the product architecture, binds their variability points according to requirements and design, and implements product specific assets. Compared to single system development, integration efforts are decreased due to predefined architectures and integration mechanisms [CN07].
- *Product Testing* ensures sufficient quality of the end product. Although the components have been tested during product line development, instantiated variability points and interaction with other components must also be covered. Furthermore, during domain testing it is impossible to cover all potential combinations of core assets.

Product development within an SPL is a recurring activity. The objects of work (i.e. products) are homogeneous and stable. It can be assumed that similar products have been developed before and that product architecture and technology are well understood. Interdependencies between the primary processes are therefore expected to be low. Compared to scenario A, product development is characterised by low dynamics and low complexity. Consequently, an activity-based decomposition of work seems most appropriate, which will result in divisional units [Gro95]. The degree of decomposition is determined by a preferably small team size [Bal08], and the required interaction between the resulting functional units. For product development, the present work suggests a structure as depicted in the following.

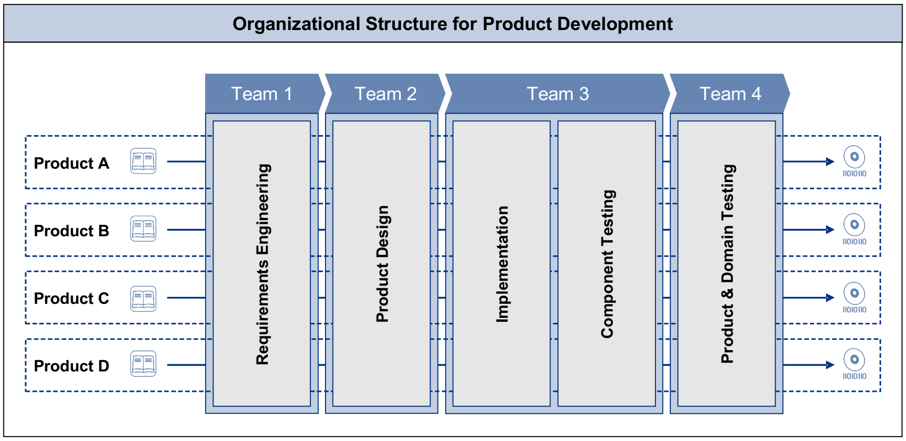

Figure 2: Organizational Structure for Product Development

The above structure assumes a low interdependency between the five primary processes of product development, except for implementation and component testing. This is due to the fact that the product line requires development to occur within clear bundaries with precisely defined interfaces between core assets and architectures. It however does not predefine requirements for component testing, which is therefore joined with the implementation process due to presumably interative interactions between both.

3 A Three Layered Structure for SPL in Systems Integration

As described before, it seems disputable whether the concept of software product lines in its original form can be applied to the field of systems integration (SI). Revisiting chapter 1, the major issues can be subsumed as follows:

- *Integration across software product line boundaries:* SPLs have a preferably narrow scope to be most powerful. SI however requires considering a variety of different products with a very different scope. If the product lines of a system integrator are not compatible with each other, integration will inevitably occur outside their boundaries.

- *Multiplicity of technologies:* The multiplicity of different technologies in SI, caused by high heterogeneity, inflexible legacy systems, and different data sources, seems to be a major drawback to distinct product lines. Too much scope, which will most likely be used only once, would have to be added.
- *Uncertain return of investment:* As systems integration produces very customer specific solutions, the minimum number of products to break even cannot be ensured. The cost for a substantial software product line may outweigh its savings.

It is assumed that an integration of different IT systems mostly occurs within the boundaries of a particular industry. An automotive supplier for instance will hardly need to integrate any of his systems with an e-government solution from the public sector. Yet he may require integrating his SAP accounting system with a logistics application of one of his suppliers. This assumption is backed by current organizational structures of major systems integration companies, which are organized in a vertical structure based on certain industries [Pie09]. Any integration architecture should therefore at least support the typical systems of the respective industry. Implementing such an architecture within an software product line however, would broaden its scope far beyond being efficient and thus feasible for industrialization. This especially applies to reusable core assets, which would be too generic to provide any benefit.

To overcome these problems, a three layered approach for software product lines in systems integration has been developed. It essentially adds a layer of abstraction on top of the software product lines. The contents of each are suggested as follows:

- The *Business Domain Layer* is a new super ordinate layer that spans over a complete division or business segment within a system integrator's organizational structure. It identifies the major requirements of the business domain in scope and conceptually defines fundamental core assets, technologies, and systems typically used therein. The development of an abstract system landscape and integration architecture ensures the interoperability of different systems and product lines within the business domain. It consists of the four core processes Business Domain Analysis, Portfolio Definition, Architecture & Roadmap Definition, and Core Asset Development.
- The *Product Line Layer* consists of several software product lines identified in the Portfolio Definition process of the Business Domain Layer. The Engineering processes of these software product lines differ only marginally. The most obvious variance to conventional software product line engineering is the lack of the Business Domain Analysis process, and a reduced Domain Requirements Engineering process. These functions are now incorporated in the Business Domain Layer and provide their findings to the subsequent product lines. All other processes remain the same but must adhere to the specifications and utilize the provided core assets from the business domain layer.
- The *Production Layer* contains the actual development of a product within a software product line. It does not differ from the conventional concept of software product lines depicted in section section 2.2. This of course only applies to the software

development process. Any systems integration specific architecture or solution design was already taken care of in the previous two layers.

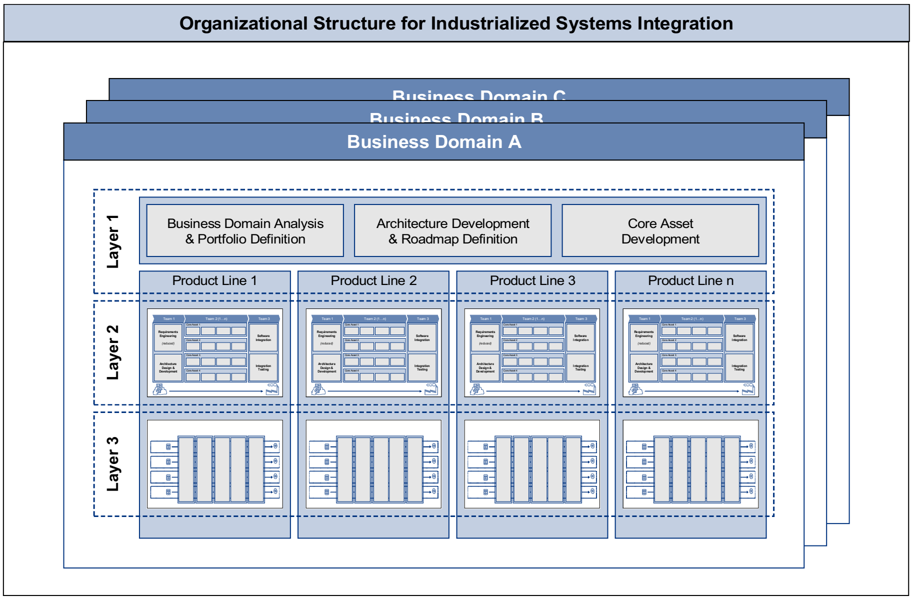
Organizational Structure for Industrialized Systems Integration
Business Domain C
Business Domain B
Business Domain A
Layer 1
Layer 2
Layer 3
Business Domain Analysis & Portfolio Definition
Architecture Development & Roadmap Definition
Core Asset Development
Product Line 1
Product Line 2
Product Line 3
Product Line n

Figure 3: Three Layered Approach for Industrialized SI

The work objects of the above processes can be combined into a product line skeleton, which will be instantiated by a particular software product line. This rather abstract layer for industrialized software development is expected to have a positive effect on the previously mentioned major issues of industrialized systems integration. The first concern, *integrating products from different product lines*, may be solved by the Portfolio Definition and Architecture & Roadmap Definition processes. The abstract architecture applicable to all software product lines ensures the compatibility of products within a given business domain. Integrating the products from a supply chain management product line with those from a shopfloor systems product line, for instance, will be much easier due to compatible architectures and technologies. The second one, *multiplicity of technologies*, can be alleviated by a joint technology roadmap. It will limit the number of utilized technologies within the software product lines and thus reduce their heterogeneity. While differences due to legacy systems or third party applications will prevail, these may be alleviated by joint interface components across multiple product lines. Within the product line boundaries, heterogeneity is thereby reduced to standardized interfaces, while externally a wide variety of products may be supported. Furthermore, if products from such product lines are integrated with each other, these issues will resolve over time. The third concern, *ensuring the return of investment*, can also be attenuated. Software product line engineering may instantiate the predefined skeleton and has a greatly reduced effort in the processes Business Domain Analysis, Business Domain Architecture, Architecture Design & Development, and Core Asset Development. Due to reduced efforts and thus cost, the break even point of a SPL may be reached earlier. Although this approach may not be as efficient as traditional software product lines, the author assumes that it still helps to advance their break even point in SI and that it will have a positive effect on the overall product quality. Case studies and real world implementations in traditional software development have shown a break even after 2–3 products of a product line [CN07]. This value of course depends on the effeciency and reusability of the underlying core assets. However, if these characteristica can also be achieved for the core assets of a systems integration product line still has to be proven in practice.

As implementing the business domain layer is a singular and novel undertaking, work is decomposed based on work objects. Thereby the processes Business Domain Analysis and Portfolio Definition are combined due to a presumably close interaction. Architecture Development and Roadmap Definition, as well as Core Asset Development remain separate as they only rely on their predecessor's outcomes but do not significantly influence them. Based on the nature of the core assets to be developed, it is also conceivable to break it down into different teams. These teams may then be responsible for particular assets throughout their lifetime and also take over their maintenance.

The resulting structure of the three layered approach is depicted in Figure 3. It should be noted that the Product Line layer does no longer contain the Business Domain Analysis Process form Software Product Line Engineering and also reduces the responsibilities of the Domain Requirements Engineering Process. These functions are now incorporated in the Business Domain Layer and provide their findings to the subsequent product lines. All other processes remain the same but must adhere to the specifications and utilize the provided core assets from the business domain layer. The internalization and thus organizational structure of the remaining product line engineering processes remains the same.

4 Further Research

The present work introduced into the fundamentals of software industrialization and systems integration, and showed that both cannot be combined easily. The reasons therefore are integration issues across product line boundaries, a high heterogeneity, and an unsure return on invest. In spite of these issues, the paper developed an alternative approach to implement software product lines as one of the industrial key concepts in the field of systems integration. As the preset work has a rather conceptual character, further research is required to move it closer to industrial practice. This especially includes developing a more detailed process and role model to give practitioners a starting point for realisation. Furthermore, the exemplary implementation of a Business Domain Layer as a proof of concept, especially with regard to abstract, reusable assets, would be helpful to promote industrialization in systems integration.

Besides implementing software product lines as the industrial key principle of specialization, standardization, systematic reuse, and automation represent the next key milestones on the way towards fully industrialized systems integration. Further research is required to identify their current representation in software engineering and eventually apply them to the field of systems integration.

References

[Bal08] Balzert, H.: Lehrbuch der Softwaretechnik. Softwaremanagement. Spektrum Akad. Verl., Heidelberg, 2008.

[CN07] Clements, P.; Northrop, L.: Software product lines. Practices and patterns. Addison-Wesley, Boston, 2007.

[Fre98] Frese, E.: Grundlagen der Organisation. Konzept, Prinzipien, Strukturen. Gabler, Wiesbaden, 1998.

[Gro95] Grochla, E.: Grundlagen der organisatorischen Gestaltung. Schäffer-Poeschel, Stuttgart, 1995.

[Lan04] Lang, C.: Organisation der Software-Entwicklung. Probleme Konzepte Lösungen. Dt. Univ.-Verl., Wiesbaden, 2004.

[Lin07] Linden, F.: Software product lines in action. The best industrial practice in product line engineering. Springer, Berlin, Heidelberg, New York, 2007.

[PBL05] Pohl, K.; Böckle, G.; Linden, F.: Software product line engineering. Foundations, principles, and techniques ; with 10 tables. Springer, Berlin, 2005.

[Pie09] Pierre Audoin Consultants: Software and IT Services Industry (SITSI) Report, Paris, 2009.

[Töp85] Töpfer, A.: Umwelt- und Benutzerfreundlichkeit von Produkten als strategische Unternehmensziele. In Marketing ZFP, 1985, 7; pp. 241–251.

[WD08] Wöhe, G.; Döring, U.: Einführung in die allgemeine Betriebswirtschaftslehre. Vahlen, München, 2008.