Proceedings of the IASTED International Conference
3rd Annual IASTED International Conference Software Engineering and Applications
October 6-8, 1999 - Scottsdale, Arizona, USA

CHARACTERIZING EVOLUTION IN PRODUCT LINE ARCHITECTURES

MIKAEL SVAHNBERG & JAN BOSCH
University of Karlskrona/Ronneby
Department of Software Engineering and Computer Science
Soft Center, S-372 25 Ronneby
+46 457-385000
[Mikael.Svahnberg|Jan.Bosch]@ipd.hk-r.se
http://www.ipd.hk-r.se/[msv|bosch]

Abstract. *Product-line architectures present an important approach to increasing software reuse and reducing development cost by sharing an architecture and set of reusable components among a family of products. However, evolution in product-line architectures is more complex than in traditional software development since new, possibly conflicting, requirements originate from the existing products in the product-line and new products that are to be incorporated. In this paper, we present a case study of product-line architecture evolution. Based on the case study, we develop categorizations for the evolution of requirements, the product-line architecture and product-line architecture components. Subsequently, we analyze and present the relations between these categorizations.*

Keywords. Product Line Architectures, Object-Oriented Frameworks, Software Reuse, Software Evolution

1 INTRODUCTION

Product-line architectures, i.e. a software architecture and set of reusable components shared by a family of products, present an important approach to increasing software reuse and reducing the cost of software development and maintenance for software companies. Different from the popular view on component-oriented programming, the components in the product-line architectures that we have studied [1][2] are large pieces of software containing up to 100 KLOC. These components are typically modeled as object-oriented frameworks that cover functionality common for the products in the product-line and support the variability required for the various products.

Software in PLA's, once developed, is subject to considerable evolution, since a constant flow of new requirements is present. The typical way to deal with this is to create two independent evolution cycles, i.e. for each product, incorporating product-specific new requirements, and for the PLA as a whole, incorporating new requirements that affect all or most of the products in the product-line.

Although evolution in product-line architectures is a complex and difficult to manage process, it is not studied much by the research community. To address this, we have performed a case study of a product-line architecture at Axis Communication AB, a swedish company selling a wide range of printer, storage, scanner and camera server products worldwide. The company has employed product-line architecture based software development since the beginning of the 1990s and uses object-oriented frameworks as the components in the product-line.

The remainder of the paper is organized as follows. In the next section, the notion of product-line architectures and the evolution process are described. Section 3 discusses the case study method, the company, the product-line architecture and the evolution of the file-system framework component through eight releases. Based on the case, we define categorizations of requirements evolution, product-line architecture evolution and component evolution in section 4. Related work is discussed in section 5 and the paper is concluded in section 6.

2 PRODUCT LINE EVOLUTION

The evolution of a product line is driven by changes in the requirements. These requirement changes can come from a number of sources, such as the market, future needs in the company, or a desire to introduce new products into the product line. Take, for example, the organization in Figure 1, where a particular business unit is driven by a number of requirements. These requirements are divided between the products that this business unit is responsible for and the general requirements on the product line as a whole. The product-line requirements can affect either the architecture of a particular framework, thus creating a change in its interface, or it can cause a change in one or more of the concrete implementations of the framework.

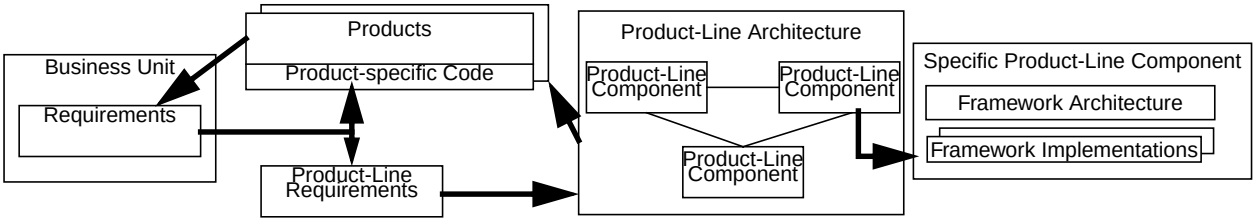

FIGURE 1. EVOLUTION OF A PRODUCT LINE

The product-line architecture with a number of concrete framework implementations is then instantiated into a set of products. The products give input as to how they, i.e. the products, should evolve, which is feedbacked into the business unit requirements, thus completing the cycle.

2.1 DEFINITION OF FRAMEWORK

As is hinted in Figure 1, we define a framework to consist of a framework architecture, and a number of concrete framework implementations. This interpretation holds well in a comparison to [3], in which a white box framework can be mapped to our framework architecture, and a black box framework consists of several of our framework implementations in addition to the framework architecture.

3 THE CASE

The main goal in our study was to find out how a framework evolves when it is part of a product line. To achieve this goal, we conducted unstructured interviews with key personnel that have been involved in the studied system for quite some time. Furthermore, we foraged their intranet for more information.

Axis Communications is a relatively large swedish software and hardware company that develops networked equipments. Starting from a single product, an IBM print-server, the product line has now grown to include a wide variety of products such as camera servers, scanner servers, CD-ROM servers, Jaz-servers, and other storage servers. Their software product line consists of 13 reusable assets in the form of object-oriented frameworks. Axis and their product line is further presented in [1] and [2].

During this study, we have focused on a particular product, the storage server. This is a product that initially allowed one to plug in CD-ROM devices onto the network. This initial product have later been evolved to several products, of which one is still a CD-ROM server, but there is also a Jaz server and recently an HardDisk server was added to the collection. Central for all these products is a file system framework that allows for uniform access to all types of storage devices.

The file system framework have existed in two distinct generations, of which the first was only intended to be included in the CD-ROM server, and was hence developed as a read-only file system. The second generation was from the beginning intended to support read-write file systems, and was implemented accordingly. Figure 2 and Figure 3 illustrates the first releases of the two generations. As is seen, generation one contains practically everything, whereas generation two is much more modularized.

3.1 GENERATION ONE

Release one. The requirements on this, the very first release of the product, were to develop a CD-ROM server. This implies support for network communication, network file system support, CD-ROM file system support, and access to the CD-ROM hardware. Requirements on the file system were to support the ISO9660 file system for CD-ROM's, and a virtual pseudo file system for control and configuration purposes. Supported network file systems were NFS and later SMB.

Release two. The goal for release two was, among other things, to create a new product, supporting Token Ring instead of Ethernet. Support for the Netware network file system was added, plus some extensions to the SMB protocol. Furthermore, the SCSI-module was redesigned, and support for multisession CD's was added. The specification of Netware was changed, so that it could be added in the same way as NFS and SMB ealier. Figure 3 summarizes these changes.

Release three. Release three of the PLA was more of a clean-up and bug-fixing project. The SCSI-driver was modified to work with a new version of the hardware, and a web interface was incorporated to browse the pseudo file system. Support for long filenames was implemented in the ISO9660-module. Figure 4 shows these changes.

Release four. Requirements for this release were to support NDS, which is another Netware protocol, and to generally improve the support for the Netware protocol. NDS required a new algorithm for acquiring access rights for a file, and all the other network file systems had to adapt the interface for this new algorithm as well. The namespace cache that was introduced in release two was removed, and the requirement was now implemented in the ISO9660-subsystem. Figure 5 summarizes these changes.

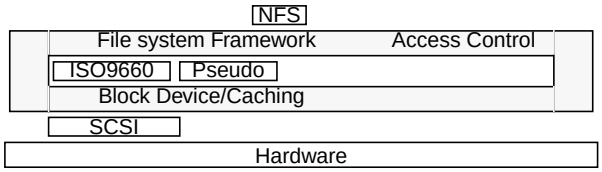

FIGURE 2. GENERATION ONE OF THE FILE SYSTEM FRAMEWORK

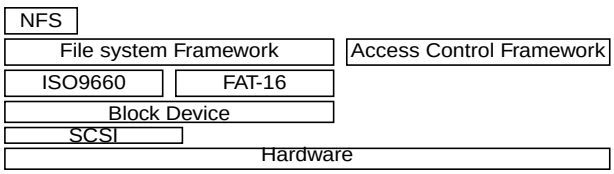

FIGURE 3. GENERATION TWO OF THE FILE SYSTEM FRAMEWORK

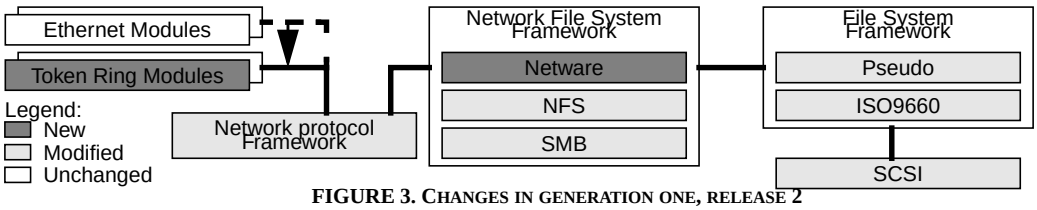

FIGURE 3. CHANGES IN GENERATION ONE, RELEASE 2

3.2 GENERATION TWO

Work on generation two of the file system framework was done in parallel to generation one for two years, as can be seen in Figure 6. After the fourth release of generation one all resources, in particular the staff, were transferred into the second generation, as is signified by the arrow between generation one and two in the picture.

Release one. Requirements on release one of this second generation were very much the same as those on release one of the first generation, with the exception that one now aimed at making a generic read-write file system from the start. As can be seen when comparing Figure 3 with Figure 2 the second generation was much more modularized from the start. NFS and SMB were supported in this first release, but with both read and write functionality. A proprietary file system, the MUPP file system was developed as a learning project.

Release two. Release two came under the same project frame as the first release, since the project scope was to keep on developing until there was a product to show; the JAZ-drive server. Added was the FAT-16 subsystem in the file system framework, and the MUPP-fs and NFS protocol developed for release one was removed. Some changes were made in the SCSI-module and the BlockDevice module. Figure 7 illustrates the changes in release two. The actual file system framework architecture remained unchanged in this release, mainly because the volatile parts were now separate frameworks.

Release three. The project requirements for release 3 were to develop a hard disk server with backup and RAID support. MTF was added to support backup tapes, and UDF was also added. An additional network management protocol, SNMP, was implemented, and Netware was copied from generation one. The access control framework was modified to work with the new network file systems. Finally, BlockDevice changed name to StorageInterface. Figure 8 depicts these changes.

Release four. The product requirements for release four was to make a CD-server working with a CD-changer. An implementation of ISO9660 was developed under the file system component, and the NFS protocol was reintroduced to the set of network file systems. Figure 9 shows these changes.

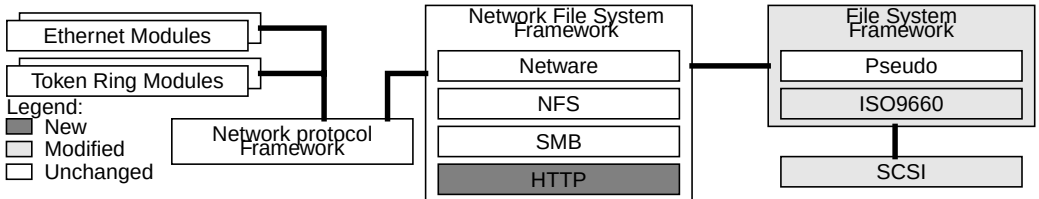

FIGURE 4. CHANGES IN GENERATION ONE, RELEASE 3

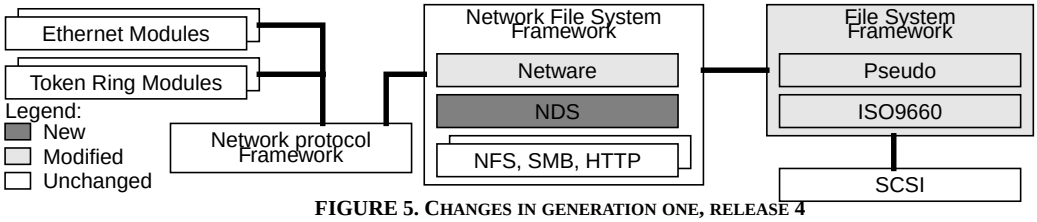

FIGURE 5. CHANGES IN GENERATION ONE, RELEASE 4

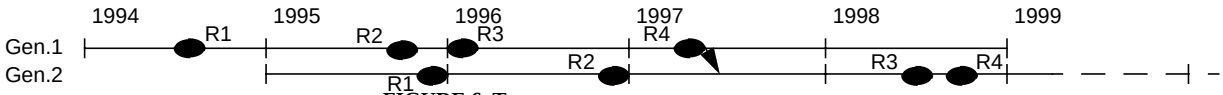

FIGURE 6. TIMELINE OF THE FILE SYSTEM FRAMEWORK

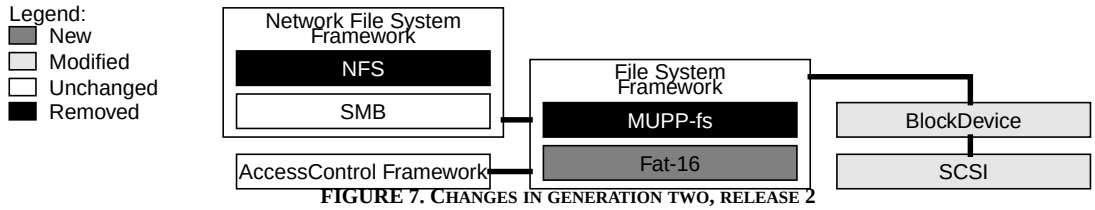

FIGURE 7. CHANGES IN GENERATION TWO, RELEASE 2

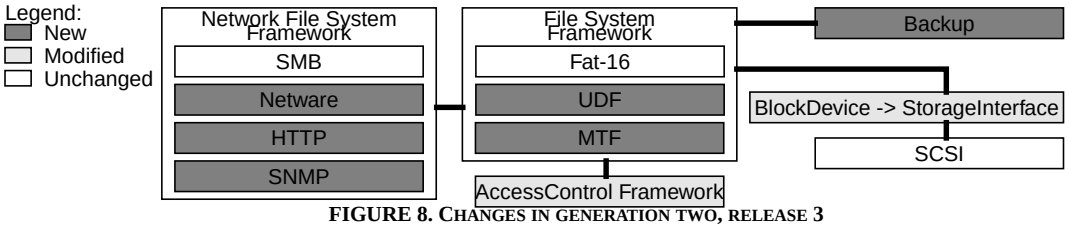

FIGURE 8. CHANGES IN GENERATION TWO, RELEASE 3

4 CATEGORIZATION

Evolution in product-line architectures is a complex and difficult to manage issue. It is important to increase our understanding of PLA evolution to improve the way we incorporate new requirements in the product-line architecture and in its components. In this section, we present categorizations of the evolution of requirements, the product-line architecture and the PLA components. Finally, based on the categorizations, we present a first version of an overall taxonomy of product-line architecture evolution.

4.1 CATEGORIES OF REQUIREMENTS

In this section, we present six categories of requirements that we found evidence of in the studied product line. Figure 10 illustrates the relations between the requirement categories further.

Construct new product family. Requirements of this kind comes when there is a need on the market for a new type of product. This generally leads to creation of new business units, new product lines, etc.

Add product to product line. Once you have a product line in place, most of the work goes in to improving the functionality, and to customize the support given to the end users. This is done by adding products to the product line that are tailored for certain needs.

Improve existing functionality. It is not enough to only create new products, you must also improve the products you have. This features supporting new standards, adding user-requested features, etc.

Extend standard support. Standards tend to be rather elaborate and, typically, only part of the full standard is implemented in new products. A typical requirement for subsequent versions is to incorporate additional parts of standards.

New version of hardware, operating system, or third party component covers more functionality. The above categories are all examples of that the software system boundaries expand to support more of the tasks that users expect of them [4]. Naturally this is also true on the levels below the system. Should a new version of the layers of functionality under the product line architecture appear, developers will naturally try to use these lower-level libraries, resulting in functionality being removed from the PLA components.

Improve quality attribute of framework/framework implementation. First versions of products often focus on providing the required functionality and, consequently, spend less effort on the quality attributes of the system.

4.2 CATEGORIES OF PLA EVOLUTION

New requirements that should be incorporated for all products have an impact on the product-line architecture. We have identified eight categories of product-line architecture evolution that may occur. Figure 11 illustrates the relations between the different categories of PLA evolution. The shaded boxes in the figure signifies that the categories enclosed have some common traits.

Split or Derived product line architecture. When it is decided that a new set of product should be developed, a decision must be taken of whether it can be kept in the same product line as a branch (Derived PLA), or whether it is necessary to split the PLA into two, meaning that one uses the existing product line as a template for creating another.

New, Changed, Split, or Replaced product line architecture component. New components are added because new products require new functionality. Components are changed in many cases, as discussed below. To split or replace a component is mainly done to improve on a quality attribute.

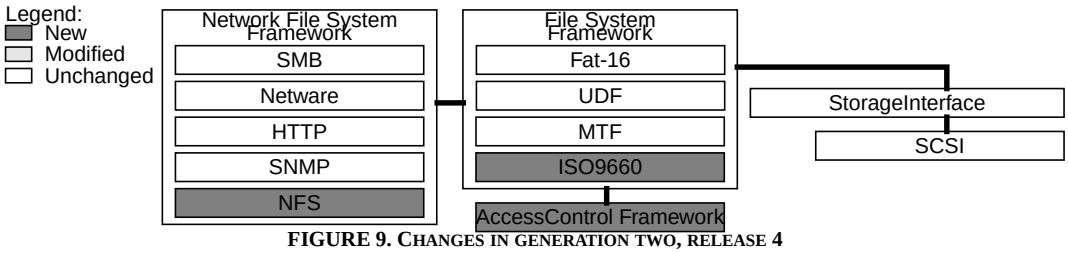

FIGURE 9. CHANGES IN GENERATION TWO, RELEASE 4

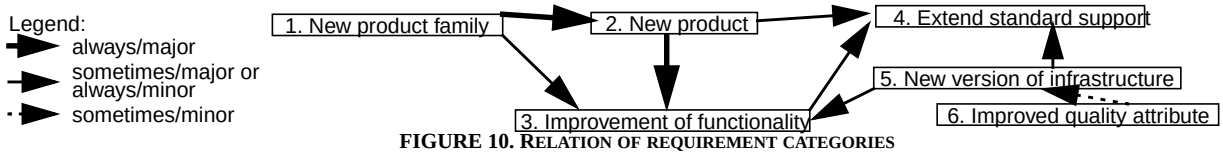

FIGURE 10. RELATION OF REQUIREMENT CATEGORIES

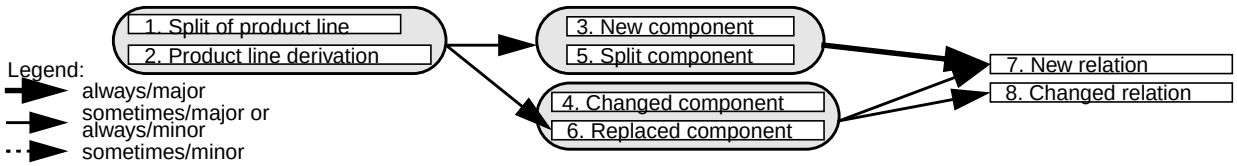

FIGURE 11. RELATION OF PRODUCT LINE ARCHITECTURE CATEGORIES

New and changed relation between components. As a consequence of the above, relations may be introduced or changed. A changed relation also includes a removed relation.

4.3 PLA COMPONENT EVOLUTION

In the previous section we discussed how the product line architecture can change. In our experience, category four, i.e. changes to a product-line architecture component, is the most common type of evolution. In this section, we discuss the different types of evolution that a component can be subject to. Figure 12 depicts how the product line architecture component evolution categories relate to the component itself.

New framework implementation. Given a framework architecture, the traditional way of extending the family of supported functionality is by adding implementations of the framework.

Changed framework implementation. Evolution of this type may occur when new functionality is added, or when existing functionality is rewritten to, for example, better support some quality attribute.

Decreased functionality in framework implementation. As a direct consequence of the requirement category concerning new versions of hardware, operating systems and other third party components, the functionality in a framework implementation can decrease, since part of its functionality is now handled elsewhere.

Increased framework functionality. This may seem similar to the first category, but they are not quite on the same level. Whereas category one adds a new implementation that supports the same functionality as its peers, this category adds functionality to all of the framework implementations. This occurs when there is a framework gap [5], i.e. that the desired functionality is not covered by any of the composed frameworks.

Add external component to add functionality with otherwise framework architectural impact. This is an interesting category, which we have found evidence of not only in our case, but also in other companies. As a project begins to draw towards its end, there is a growing reluctance to go in and meddle with the underlying frameworks, since this yields more tests, and a higher risk for introducing bugs that have impact on a large part of the system. Then a new requirement is added (or an existing requirement is reinterpreted), leading to changes with architectural impact, if implemented normally. To avoid the excessive amounts of effort required to incorporate the architectural changes, one instead develops a solution that implements the requirement outside of the framework.

4.4 RELATION OF CATEGORIZATIONS

In the previous sections, we have presented categorizations for the evolution of requirements, the PLA and PLA components. However, these are not independent, but can be related to each other. Figure 13 shows how the requirement categories lead to different types of evolution both on the PLA and on the PLA components. As can be seen, many of the evolution types are not directly caused by requirements, but are indirect effects of the requirements, in that one change triggers another change of a different type.

5 RELATED WORK

Most categorizations of requirements found are rather high-level. For example, Tracz discusses reference requirements [7] on domain-specific software architectures. Prieto-Diaz [8] simply divides requirements into the categories of stable and variable requirements.

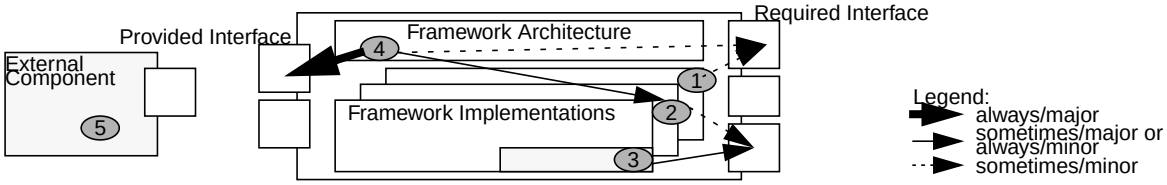

FIGURE 12. CATEGORIES OF PRODUCT LINE ARCHITECTURE COMPONENT EVOLUTION

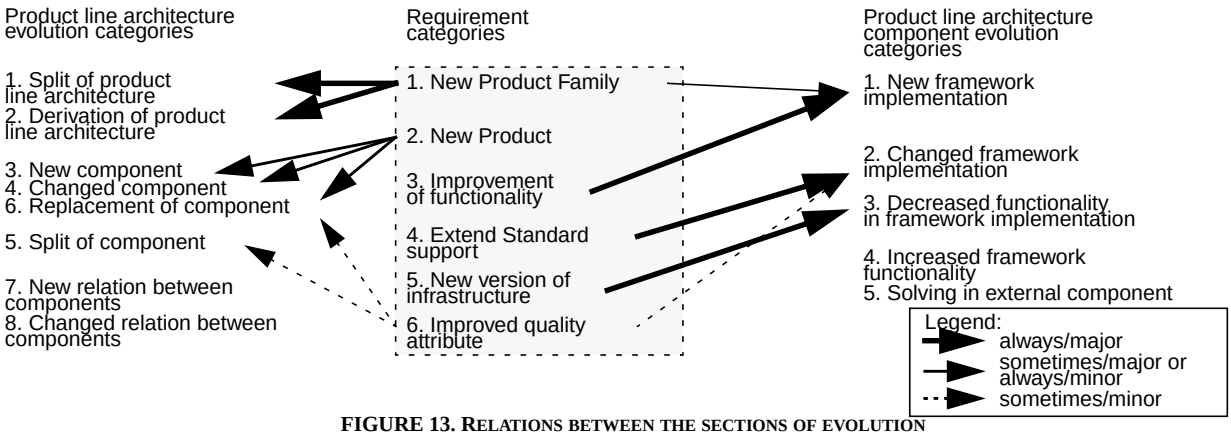

FIGURE 13. RELATIONS BETWEEN THE SECTIONS OF EVOLUTION

With respect to characterizing evolution, some related work exists. Siy & Perry describes the management of a large product line architecture [9], but their focus is primarily on configuration management rather than on the requirement and framework evolution. Lehman [10] states a number of laws concerning evolution of software, but these are too fundamental, and also too high-level to be applicable to our work.

A paper by Mattson and Bosch describes how frameworks evolve, and the types of changes that a framework can become subject to [11].

FODA [12] is a method for domain analysis that is developed by the Software Engineering Institute. A part of FODA is to construct graphs of features, in order to spot commonalities and benefit from these commonalities when planning reuse and adding new features. Much effort is put into the analysis model, to foresee future enhancements on the features or components. Comparing FODA to our work, we see that FODA focuses on the spotting of commonalities by constructing domain-specific feature categories, and the relations between the categories. The categories we have presented are not based on particular features, but rather on possible steps of evolution. In that sense, FODA is orthogonal to our work. Related to FODA is FeatureRSEB [13], who extends the use-case modeling of RSEB [14] with the feature model of FODA.

6 CONCLUSIONS

Product-line architectures present an important approach to increasing software reuse and reducing development cost by sharing an architecture and set of reusable components among a family of products. However, evolution in product-line architectures is more complex than in traditional software development since new, possibly conflicting, requirements originate from the products in the product-line and new products that are to be incorporated.

Based on a case study, we present categorizations of the evolution of the requirements, the PLA and the PLA components. Using the categorizations, we relate the three levels of evolution to each other. The result is a first version of a taxonomy of product-line architecture evolution.

By creating this taxonomy, we hope to have improved understanding of how a framework inside a product line architecture evolve as a consequence of direct requirements on the framework, and indirect requirements on other parts of the product line architecture. This improved understanding helps, we believe, in being able to on forehand understand the impact of a new requirement. This, in turn, helps us to avoid an overly rapid aging product line architecture, i.e. architecture erosion.

We intend to continue working on the taxonomy by conducting more case studies to confirm the findings and to investigate the relations further. In so doing, we hope to achieve a prediction model presenting the changes one can expect as a consequence of a particular requirement.

REFERENCES

[1] J. Bosch, Evolution and Composition of Reusable Assets in Product-Line Architectures: A Case Study, *Proc. of the First Working IFIP Conf. on Software Architecture*, February 1999.

[2] J. Bosch, Product-Line Architectures in Industry: A Case Study, *Proc. of the 21st Int'l Conf. on Software Engineering*, May 1999.

[3] D. Roberts, R.E. Johnson, Evolving Frameworks: A Pattern Language for Developing Object-Oriented Frameworks, *Proceedings of PLoP - 3*, 1996.

[4] J.L. Marciniak (Editor), *Encyclopedia of Software Engineering*, Lehman, M.M., on "Software Evolution", New York: John Wiley & Sons, 1994.

[5] M. Mattsson, J. Bosch, "Framework Composition: Problems, Causes and Solutions", In *"Building Application Frameworks: Object Oriented Foundations of Framework Design"* Eds: M.E. Fayad, D.C. Schmidt, R. E. Johnson, New York: Wiley & Sons, ISBN#: 0-471-24875-4, Forthcoming.

[6] F.P. Brooks, *The Mythical Man Month* (anniversary edition), New York: Addison-Wesley, 1995.

[7] W. Tracz, DSSA (domain-specific software architecture): pedagogical example, *SIGSOFT Software-Engineering Notes*. vol.20, no.3; July 1995; p.49-62

[8] R. Prieto-Diaz, Reuse Library Process Model, Technical Report AD-B157091, IBM CDRL 03041-002, STARS, July 1991.

[9] H.P. Siy, D.E. Perry, Challenges in Evolving a Large Scale Software Product. Principles of Software Evolution Workshop. 1998 International Software Engineering Conference (ICSE98), Kyoto Japan, April 1998.

[10] M.M. Lehman, On understanding laws, evolution and conservation in the large program life cycle, *Journal of Systems and Software*, 1(3):213-221, 1980.

[11] M. Mattsson, J. Bosch, Frameworks as Components: A Classification of Framework Evolution, *Proceedings of NWPER´98 Nordic Workshop on Programming Environment Research*, Ronneby, Sweden, August 1998.

[12] K. Kang et al, Feature-Oriented Domain Analysis Feasibility Study, SEI Technical Report CMU/SEI-90-TR-21, November 1990.

[13] M.L. Griss, J. Favaro, M. d'Alessandro, Integrating feature modeling with the RSEB, *Proc. of the Fifth Int'l Conf. on Software Reuse*, IEEE Computer Society, Los Alamitos, CA, USA, p.76-85., 1998.

[14] I. Jacobson, M. Griss, P. Jonsson, *"Software Reuse: Architecture, Process, and Organization for Business Success"*, New York: Addison-Wesley-Longman, May 1997.