A Software Product Line Experience

Natalia Andriano
GSG Argentina
natalia.andriano@motorola.com

Jorge Kirch
GSG Argentina
jorge.kirch@motorola.com

Mariano Zibecchi
GSG Argentina
mariano.zibecchi@motorola.com

Abstract

Previous work in software product lines showed an increment of up to 10 times in productivity and product quality [SEI Prod_Line]; because of this, the software product line approach is critical for the success of a development project including a common platform. There are several differences between the traditional development techniques and the software product line approach. This initiative differs in scope and approach that is the reason why new areas of study were created, namely Domain Engineering and Software Product Line Architecture.

For the development of a software platform in GSG Argentina we took the software product line approach and combined different known practices of Domain Engineering and Software Architecture in order to adapt them to our project's context.

This paper describes lessons learned during the development of this project, such as the tailorization of the application of FORM (Feature-Oriented Reuse Method), the use of existing standards and cases of study to extract and infer knowledge of the problem domain, how we interact with marketing and system engineering areas in order to gather relevant functional and quality features of the platform, documentation practices, challenges we found and the approaches we took to solve them.

Key Words*: Software Product Lines, FORM, Core Platform, Domain Engineering, FODA, Feature Model, Application Engineering.*

Workshop Name: *Ingeniería de Software y Bases de Datos (WISBD)*

1. Introduction

Product line software engineering is an emerging paradigm that helps organizations to develop their wares from reusable core assets rather than from scratch. To develop these assets engineers must exploit commonality and manage variability. Previous work in software product lines showed an increment of up to 10 times in productivity and product quality *[SEI Prod_Line]*; because of this, the development of software product line is critical for the success of a development project including a common platform.

For the development of a software platform in GSG Argentina we decided to tailor the FORM methodology and took the software product line approach and combined different known practices of Domain Engineering and Software Architecture in order to adapt them to our project's context.

The software product line initiative is different enough in scope and approach from usual and traditional process development techniques, that is why new areas of study were created, namely Domain Engineering and Software Product Line Architecture.

One of the fundamental practices in the Product Line Software Engineering area is Domain Engineering (Core Asset Development) *[SEI Prod_Line]*. Within this area, emphasis is given in the identification and development of reusable assets from an application "domain" perspective.

One of the main Domain Engineering practices existing is the FODA (Feature-Oriented Domain Analysis) methodology. FODA was founded on a set of modeling concepts and primitives used to develop domain products that are generic and widely applicable within a domain *[SEI Dom _Eng]*. The basic modeling concepts are abstraction and refinement. Abstraction is used to create domain products from the specific applications in the domain. These generic domain products abstract the functionality and designs of the applications in a domain *[SEI Dom _Eng]*. The generic nature of the domain products is created by abstracting away "factors" that make one application different from other related applications. The FODA method advocates that applications in the domain should be abstracted to the level where no differences exist between the applications *[SEI Dom _Eng]*.

As FORM (Feature-Oriented Reuse Method) extends FODA, we decided to tailor this methodology as it covers a lot more of the development process. FORM is a systematic method that looks for and captures commonalities and differences of applications in domain in terms of "features" and using the analysis results to develop domain architectures and components *[Kang98]*. The main output of this method is the Feature Model; this model is used to support both engineering of reusable domain artifacts and development of applications using the domain artifacts. As we said before, FORM extends FODA to the software design phase and prescribes how the Feature Model is used to develop domain architectures and components for reuse. The core of FORM lies in the analysis of domain features and use of these features to develop reusable domain artifacts *[Kang98]*.

This paper describes our tailorization experience of the FORM methodology and our experience in the application of several software product line practices. It also summarizes the difficulties we had to walk through and lessons learned.

This paper is organized in five sections: section 2 displays the project context, its characteristics and problems found. Section 3 describes all the alternative tools used in order to manage the project. Section 4 presents all lessons learned during the tailorization of the method; section 5 encompasses the conclusion.

2. Project Context

GSG Argentina works as a provider of software development services to internal and external customers. GSG Argentina has been assessed as a CMM level 5 organization in 2004.

This paper is based on the experience we obtained from a software development project in Motorola Argentina Center of Software (MACS). The main objective of the project is to develop a number of asset visibility (DEFINE) solutions based on a common software platform. The solution implementations are built on top of the core platform. In some cases

where a set of solutions are closely related, an intermediate layer to cover the corresponding vertical market is built based on the core platform. Any solution can be seen divided in three different layers, going from the specific solution to the more generic core: solution implementation, vertical market platform and core platform.

The customer of the project is a new Motorola organization specialized in asset visibility solutions, and MACS works as its only software engineering force. The customer is composed of three major teams: marketing, product management, and engineering.

Given the asset visibility problem domain this organization is targeted at seemed to be quite well defined, we found the project context appropriate for a Software Product Line (SPL) approach []. Nevertheless, one of the main difficulties that we had to deal with this approach was the limited information the customer representatives provided to us regarding the problem domain. This was specially evident in the identification and definition of non-functional requirements. Meetings with the customer are performed regularly in order to gather and refine requirements, and we also used research papers from specialized consulting firms on related markets to obtain insight on the asset visibility problem domain.

Regarding the development team, it is geographically distributed in two groups located in Argentina and USA, with a time zone difference relatively small (2-3 hours). The team is composed of 65 engineers (including management) structured in four teams: Requirements, Architecture, Development and Testing.

The Requirements Team is responsible for eliciting, analysis and specification of system requirements; it's also responsible for the development of the feature list, the Feature Model and the Software Requirements Specification (SRS).

The Architecture Team is responsible for converting the requirements into a specification of the software product and communicate it effectively to developers, integration testers and project stakeholders. It also maintains the conceptual coherency of the software product, manages technical risks of the project related to the software,and serves as point of reference for software related issues to managers, marketing, requirement engineers and other engineering groups.

The Development Team is responsible of low level design, coding and unit testing. The Testing Team is responsible for the development of the test artifacts (test cases and internally developed test tools) and the execution of integration and system test. An independent verification and validation team is responsible of the customer acceptance tests.

The project is being developed following an iterative life cycle model, where development is divided in several releases. Each one of the releases has one or more iterations, internally called "drops". Maintenance of the software solutions are going to be faced as new projects.

3. Alternatives/Tools

Considering the characteristics of our project (several solutions, based on the same core platform) our objective is to reuse as much as possible, not just code but other core assets too. Then, a primary approach was obtained from the SEI Software Product Line Initiative *[Clements01]*, due to all the organizational benefits that it provides, such as: large-scale

productivity gains, improvement of time-to-market, improvement of product quality, increase of customer satisfaction, achieve reusable goals, among others *[Clements01]*.

We decided to tailor the FORM methodology and added some probed practices based on our predefined process and then compared it with the FORM method within the product line asset development process.

FORM is a systematic method that looks for and captures commonalities and differences of applications in domain in terms of "features" and using the analysis results to develop domain architectures and components *[Kang98]*. FORM is divided between Domain Engineering Activities (Core Asset Development) and Application Engineering Activities (Product Development). The purpose of the Domain Engineering Activities is to develop domain artifacts that may be used (and reused) in developing applications for a given domain *[Kang98]*. Consist of activities for gathering and representing information on systems that shape a common set of capabilities and data. It is composed of three phases: Context Analysis, Domain Modeling and Architecture Modeling *[Kang98]*. The purpose of the Application Engineering Activities is to develop a specific application making use of the domain knowledge obtained during Domain Engineering *[Kang98]*.

After a comparative analysis of several technologies, we decided that the project shall be developed in Java. This decision was based on the result of a benchmarking between different technologies, historical data about the productivity that the centre maintains, the level of skills that all people that participates in the project have and some own characteristics of the language such as multiplatform.

The Development Team is divided in two: Domain Engineering Team and Application Engineering Team. The Domain Engineering Team has already performed the following activities regarding the Context Analysis phase defined by FORM: context analysis where the scope of the domain was defined, the relationships to other domains were identified and the inputs/outputs of the domain were specified.

Regarding the Domain Modeling phase defined by FORM, our customer's Product Management team created, based on Marketing Plans, a Feature List (not prescribed by the methodology) in order to identify the main functionality that the domain should support. This initial Feature List was created by defining all high level features identified as needed by the current solutions.

Then two teams worked in parallel to elicit requirements based on the feature lists, one for the core platform and another one for the solutions. From that elicitation work, feedback was obtained and introduced into the Feature List and thus into the Feature Model to complete and refine the required features for the core platform and each software solution.

Then the Feature List was taken as an input for the development of a Feature Model, where all features were divided between layers (capability, operating environments, domain technology and implementation techniques) and identified as optional, mandatory or alternatives, as FORM prescribes. At the end of this process – which was iterative- we ended up with two Feature Lists: one for the core platform, and another one for each particular solution.

As stated in *[SEI_Dom_Eng]* the informational analysis captures and defines the domain knowledge and data requirements that are essential for implementing applications in a domain. The purpose of this analysis is to represent the domain knowledge explicitly in terms of domain entities and their relationships, and to make them available for the derivation of objects and data definitions during the operational analysis and architecture modeling. It also defines data that is assumed to come from external sources. The result of this analysis is the Informational Model. We decided not to develop this model since our Feature List specifies the essential data required for the core platform and each software application and the relationships between them and defines explicitly the source of every external data and the priority that each defined features has.

The Operational analysis identifies the control and data flow commonalities and differences of the applications in a domain. This activity abstracts and then structures the common functions found in a domain and the sequencing of those actions into the Operational Model *[SEI_Dom_Eng]*. On the other hand, we found that we covered the Operational Model purpose through the development of a Software Requirement Specification document already defined by our own process, so we did not develop this model either.

We found that our Feature List was much more useful than the development of an Informational Analysis and the Operational Analysis prescribed by the methodology, since it helped us with the communication, understanding and validation with the stakeholders of the functionality that the system (core platform and each software solution) shall provide; we saw that they could be replaced by this list without losing any significant information.

An example of the Feature Model developed regarding the Operating Environment layer is shown in *Figure 1: Operating Environment*:

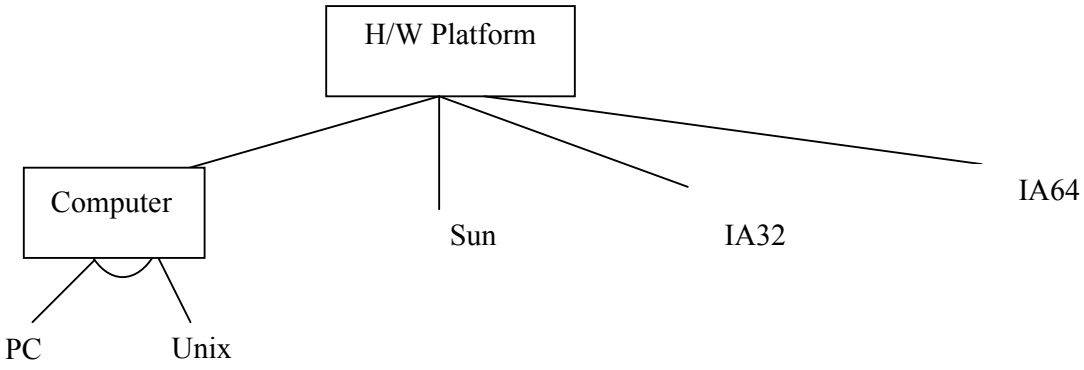

Figure 1 – Operating Environment

A domain dictionary was also created in order to maintain a common language between the stakeholders *[Kang90]*, identifying terms that are used differently or in a specific way within the domain.

The iterative nature of the FORM product Line Engineering process is also described in *[Kang02]* as shown in *Figure 2 – FORM Product Line Engineering Process.*

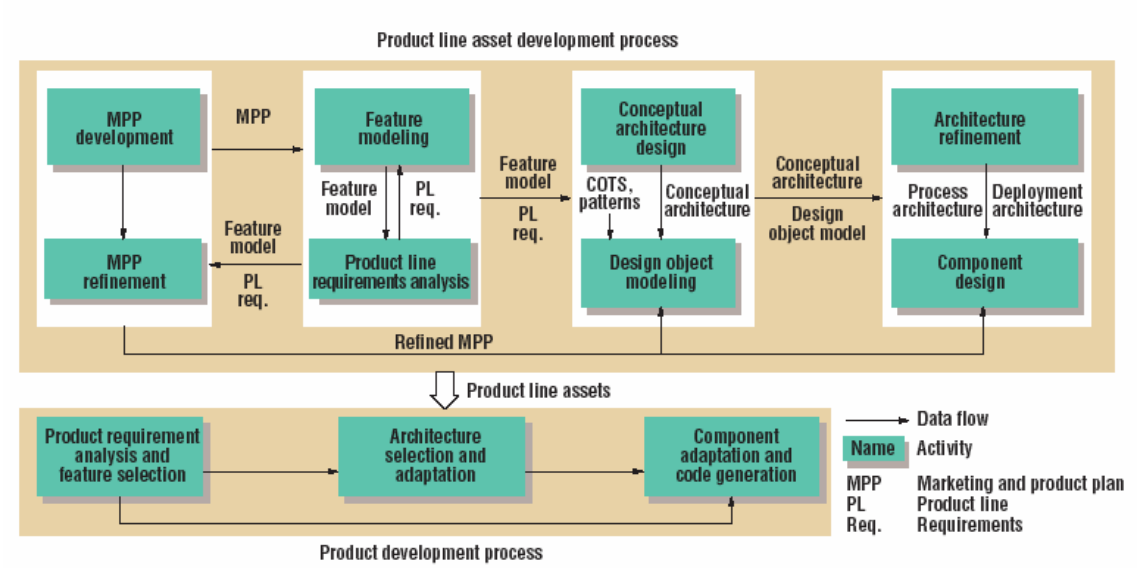

Figure 2 - FORM Product Line Engineering Process [Kang02]

Once the common features and the non common features were identified and separated, a conceptual architecture was specified for the core platform identifying appropriate extension points to provide the core with the extensibility mechanisms needed to adapt it easily to different solutions. The Software Product Line approach, FODA and FORM practices stand out the importance of a good architecture. The architecture is highly important because it represents earliest design decisions which are the hardest to change, the most difficult to get right and is also the communication vehicle among stakeholders *[SPL]*.

4. Lessons Learned

At the moment of this writing, three iterations of the core platform and a solution were developed in order to manage technical risks and to validate non-functional requirements. The last release implemented the most important non-functional requirements we identified so far for the core platform (performance, network usage and capacity), and much of the essential functionality of the solution. During these iterations we identified some problems and also some good practices that helped us in resolving these problems. From this experience we can draw some conclusions regarding our tailored FORM approach.

We found the Feature Model as a very useful tool for representing and discovering commonalities / variations between applications within the same problem domain. The Feature Model is easy to understand by the customer since it is a model close to the language marketing people uses. The organization in layers proposed by FORM was of a great influence because it helped us in the discussion of some features with our customer. Composed-by or Implemented-by relationships are also powerful mechanisms to refine the features.

As stated in *[Clements01]*, core assets are not just reusable code. We developed other important reusable assets. As an example, we can mention the development of our own test automation tool. This tool was, from the beginning, intended to be reusable across all solutions, and includes in its design common features required by them. Specific needs of

each solution are covered via configuration changes. This approach allows us to be able to quickly setup a completely automated test environment for a different solution, without any code changes. All is needed are the scripts that implement the test cases for that specific solution.

Another topic is requirements reuse. As stated in *[Kuloor]*, it is essential to have a requirements engineering process that considers a software product line approach. Also, Larn *[Larn]* enumerates the benefits of requirements reuse. In our case, work was done in order to write solution specific requirements as "deltas" from the common set, since solutions shares a subset of common requirements (i.e. core platform requirements). This work also included the study of adaptations needed in our requirements management tool, to support this.

We perform review meetings with the Testing Team after the system requirements are baselined; these meetings showed up that we needed to improve the communication channels between the different teams, due to some inconsistencies found between the Feature Model and the Architecture.

Another activity to mitigate the risk of communication problems and inconsistencies between the Feature Model and the architecture is that the Architecture Team is participating in requirements' analysis and specification and in the review activities. The Architecture Team is more aware of the problems that might appear and are able to perform a feasibility analysis of the requirements that are written.

It was difficult to elicit non functional requirements for the platform. To solve that problem, an iterative approach was taken: a selected subset of quality requirements were allocated to each iteration according to their importance and technical risk exposure associated to them. This approach helped us to address early those non-functional requirements that may impact heavily on the architecture of the system.

5. Conclusions and Future Work

This project imposes us many challenges. The development of a core platform and different solutions at the same time force us to change some of the traditional techniques used at the center, specially those related with the early stages of the development like requirements and architecture.

Our tailored approach to FORM help us to solve many problems. In the requirements area, we found that the development of a Feature Model, was of great help for eliciting functional requirements and for communicating and validating those requirements with the stakeholders. It is a valuable tool, that can be easily understood and adapted by almost all the stakeholders.

We still need to continue working in requirements reuse. The set of tools we use to manage requirements needs to be tailored to maximize reuse between different solutions/domains. Requirements change management in this context is also an area that deserves more analysis.

From point of view of the architectural, we found that the generic architecture developed is scalable and the operational cost (deployment and management) is really low. This approach

has given us good cost/features ratio, due to the great amount of opportunities that assets reuse and generic architecture represents.

The Software Product Line approach is a powerful tool. It help us to organize and to structure our teams, gave us many tools to deal with the complexities our project and provide us with a guideline to continue improving our work

Acknowledgments

We wish to acknowledge the constructive comments of Diego Rubio, Alvaro Ruiz de Mendarozqueta and Patricio Maller.

References

[Clements01] *Software Product Lines: Practices and Patterns*, Paul Clements, Linda Northrop. Addison-Wesley, 2001.

[Kang02] *Feature-Oriented Product Line Engineering*. K.C. Kang, J. Lee. IEEE Software 2002.

[Lee02] *Concepts and Guidelines of Feature Modeling for Product Line Software Engineering*. K. Lee, K. C. Kang, J. Lee. Springer-Verlag 2002.

[Kang90] *Feature-Oriented Domain Analysis Feasibility Study*. Kyo C. Kang; Sholom G. Cohen; James A. Hess; William E. Novak; A. Spencer Peterson (CMU/SEI-90-TR-21, ADA235785). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1990.

[Kang98] *A Feature-Oriented Reuse Method with Domain-specific Reference Architecture*. yo C. Kang, Sajoong Kim, Jaejoon Lee1, Kijoo Kim, Gerard Jounghyun Kim, Euiseob Shin2 Department of Computer Science and Engineering Pohang University of Science and Technology (POSTECH) San 31 Pohang, Kyoungbuk 790-784, Korea

[SEI_Dom_Eng] Software Engineering Institute. http://www.sei.cmu.edu/domain-engineering/domain_eng.html

[SPL] *Software Product Lines*. Paul Clements, Linda Northrop. Product Line Systems Program.

[Kuloor] *Requirements Engineering for Software Product Lines*. Chethana Kuloor, Armin Eberlein. The University of Calgary, Canada.

[Larn] *Ten Steps Towards Systematic Requirements Reuse*. W. Larm, J.A. McDermid, A.J. Vickers. The University of York.

[SEI Prod_Line] http://www.sei.cmu.edu/programs/pls/sw-product-lines_05_03.pdf