Building a Requirements Engineering Methodology for Software Product Lines

Tom Huysegoms, Monique Snoeck, Guido Dedene

Faculty of Business and Economics, Management Information Systems Group, K.U.Leuven
Naamsestraat 69, 3000 Leuven, Belgium
{Tom.Huysegoms, Monique.Snoeck, Guido.Dedene}@econ.kuleuven.be

Abstract. Software product lines are a great way to achieve reusability when they are correctly implemented. Theories about the product line paradigm already exist for multiple decades, but empirical research and reports of real life success stories are still scarce. Companies often still struggle to implement a software product line, because they don't possess the necessary knowledge and therefore do not sufficiently focus on the most basic aspect of a product line, namely the variability. Variability is the key to systematic and successful reuse, and should be considered as soon as possible in any software engineering project. The goal of the research is to develop a methodology for dealing explicitly with variability in software product lines during requirements engineering, because its impact will be maximal during this phase of software engineering. The methodology will be developed based on case-study research, in order to ensure practical relevance.

Keywords: Variability, requirements engineering, case-study research, harmonization.

1 Introduction

The financial crisis has intensified the strive for cost reduction. The software product line engineering (SPLE) paradigm offers a means to achieve this cost reduction inside the domain of software engineering through the systematic reuse of product line components. SPLE has already been studied for over two decades, software product lines are first mentioned in 1990 by Kang et al. as a part of their FODA specification [1]. A good definition of this software product line concept can be found in the work of Clements et al. [2]: "A software product line is a set of software-intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way". The SPLE paradigm requires a shift away from thinking about system development on a system by system basis as in traditional software engineering towards thinking in terms of development of product families. A product family is a somehow related group of software systems. SPLE focuses on these

groups of related, highly alike software applications, and the concept of variability is used to situate the differences between these software applications.

Variability is one of the key concepts in SPLE. Without variability, the SPLE paradigm would not be capable to achieve a high level of cost reduction. The variability in the software product line allows the product line to be reused in similar, yet slightly different contexts. Although the exact same software system could be used in many different contexts, this would yield a suboptimal solution. Most of the time the software system would not fit its context perfectly, and additional efforts would be required to use it. In this way the cost benefits achieved by the reuse of a single application will be (partly) annihilated by the suboptimal results of the different implementations of the single system.

Despite the fact that variability is of utmost importance to the successful use of the SPLE paradigm, the absence of explicit reasoning about variability seems to be one of the reasons why a software product line project fails in practice. This observation was made in the exploratory study of some projects at an international bank and insurance company. The lack of a clear focus on variability during the early phases of SPLE doomed the projects even before they were fully implemented. The reason behind the fact that variability is overlooked in practice is not clear, but awareness about variability must be created in order to improve the results of any future SPLE projects. This is why we want to develop a methodology about the variability concept in SPLE, so that when a software product line is created, it will have a chance to succeed.

The paper consists of different parts. Section 2 defines the problem we wish to address and investigates related work. In section 3 we define the research plan and the type of research. In the sections 4 to 7 we then subsequently present the different aspects of the framework we wish to develop our methodology upon. Section 8 gives a summary and a conclusion, along with the direction of the future research.

2 Problem statement and related work

When we look at the state of the art literature of software engineering in general and requirements engineering more specifically, we establish the fact that there is a tendency to focus on the more theoretical aspects of research in favor of the empirical work [3]. This shortage in empirically based research (like e.g. [4]) can be one of the reasons why it is so difficult for companies to implement the theories developed in software engineering. It is not so that research results are unavailable or incomplete, but the fact that it is (perceived) difficult to use the developed theories in practice leads to reluctance to start up any new software engineering program or project, like for instance SPLE projects. This reluctance can be strengthened by the fact that most new software engineering projects require additional investment funds to be gathered. In the SPLE domain assets need to be developed and this generates extra costs at startup, while benefits are only achieved when the assets are reused multiple times (at least more than once). The benefits are uncertain and are located in the future, while the costs are certain at the starting point. In combination with the fact that the development and implementation of a software product line is perceived as difficult, implementing such a line is not an obvious choice.

Once the initial investment decision is taken and the company commits itself to the SPLE paradigm, the product line itself is created and implemented. Also then it is likely that problems will arise. Due to the lack of sufficient empirical guidelines like [5] and [6], many of the problems a company can possibly face during the creation or implementation are not well documented. Without any source of information on how to solve practical problems, the implementation of the product line is hampered at best. In the worst case, the problems that arise make the companies put a stop to their efforts in applying the SPLE paradigm and return back to the single system engineering approach. One could argue that because SPLE already exists for a long time, there ought to be solutions somewhere in the large amount of literature about it created over the years. The fact is however that many companies cannot rely on someone who is very proficient in SPLE literature, otherwise they would not be having problems with SPLE in the first place. For people who are less proficient in SPLE in particular or in software engineering in general, the complicated literature does not offer a suitable ready-to-use solution which is efficient but at the same time easy enough to grasp by anyone less than an expert.

The goal of our research is to provide the much needed bridge between the theory of SPLE and its implementation in practice [7] by offering a methodology that is usable for and encompasses all stakeholders who have a role in SPLE, from IT staff to business managers. At the center of this methodology there is the concept of variability [8], as this is the foremost driver of a successful SPLE [9]. Decisions about variability need to be taken with care, as they can impede the reuse of the software assets later on. In order to minimize the impact of implicit decisions, it is best to start thinking about variability as early as during the requirements engineering. Therefore the methodology will focus itself mainly on the requirements engineering phase of the software engineering process. How we are planning to create the methodology in concrete is explained more in detail in the next section.

3 Research method and plan

In order to ensure the practicality of our Variability enabling Software Product line and Requirements Engineering Methodology (VeSPREM) we create VeSPREM based on empirical research in the form of case studies. Starting from a theoretically grounded framework which provides the basic elements that are necessary to reason about variability, we fill in this framework based on multiple case studies conducted in two software intensive multinationals. These multinationals either develop their own information system in house (internal software provider model) or develop software in order to sell it to other companies (external software provider model). The research will focus first on the internal software provider model, because the context of the internal software provider model is likely to be more controllable than the context of the external software provider model. Most of the time the stakeholders in an internal software provider context are well known while in the external model future customers are unknown at design time. The size of the cases is also likely to be smaller in an internal context because the number of possible customers is limited to the number of internal departments who may request for software systems.

In current research [10] case study research is becoming more and more popular and is considered as a full-fledged research method. It is a qualitative research method, based on real life cases. The big advantage of case study research is that it is always based on the analysis and interpretation of empirical evidence, and therefore the link with practice is ensured. Other research methods, for instance ethnographic studies, have the same link with real life, but they are less usable in our research context because they require such long periods of time to be successfully conducted that conducting several case-studies (which we need) within a reasonable amount of time is too labor intensive for a single researcher. A drawback to case study research is claimed to be the lack of possibilities to quantitatively analyze the results of the research. The lack of a formal way to test the developed hypotheses can seem a drawback, but there are possibilities to ensure the validity of the research. Multiple case studies can be conducted, in order to create a form of replication. Yin [11] describes two forms of replication: literal replication and theoretical replication. Literal replications predict similar results in order to strengthen the conclusions from the case studies. Theoretical replications predict other results than the original cases but in such a way that the differences are anticipated. In our research literal replication will be done by conducting multiple case studies in an internal software provider context, while theoretical replications will be done by studying cases in an external software provider context (in contrast to the internal software provider context).

To start with, an initial theory needs to be developed in order to achieve the right focus for the case studies [11]. In our research, this starting theory takes the form of a framework that will then be systematically enriched with the results of the case studies in order to create a complete methodology. The initial framework that we will use as starting point of our research is presented in the following sections (section 4 to section 7). It combines the results of a survey of existing theoretical research and an exploratory observational case study in a multinational banking and insurance company. The framework represents the basic ideas and assumptions about variability in SPLE taken from literature, and extends them with findings from the observational case study. The planned case studies will provide substantial material to enrich this framework with the necessary details, so that the framework can grow organically into a full-fledged methodology. A feedback loop will then trigger a revision of any previous cases against the altered/extended framework. By forming a feedback loop the construct validity of the research will be guarded. The next four sections represent the pillars of the framework: the structure of the IT department providing the product line, the decision process concerning variability in requirements, the representation of variability in goals, features and architectural specifications and the engineering process in SPLE as a whole.

4 An multi-organizational context for IT

SPLE can be applied in many contexts. The methodology we will develop aims at developing management information systems for software intensive organizations. This setting is very different from a setting of e.g. developing a product line for embedded software for cars or cell-phones. The need for variability arises from the mul-

tinational, multi-divisional or multi-institutional property of the organization (a 'multi-' organization or 'multi-' company). Each business unit has very similar but nevertheless also very particular needs. As a result of this, the IT department needs both centralization and decentralization.

Central guidance is needed in order to obtain an overview of the total IT portfolio. This overview is needed if opportunities for reuse are to be spotted. These opportunities are marked by a significant overlap of functionality between different applications. This overlap is what is called the commonality of a product family.

Simultaneously, the decentralized view on the IT department is required to cope with the variability issues of the SPLE. Each of the geographically (or logically) divided regions has its own specific properties and domain assets need to be adjusted accordingly in order to obtain a usable implementation. The combination of both centralization and decentralization can be achieved if we assume that the IT department is structured according to the blueprint described below. The proposed blueprint is based on the Integrated Architecture Framework (IAF) of Capgemini and the Asset and Solution Framework (ASF) of KBC ICT Services studied in the exploratory case study. The proposed IT blueprint is described in Fig. 1 and represents the structure for the whole IT department of a software intensive 'multi-'company.

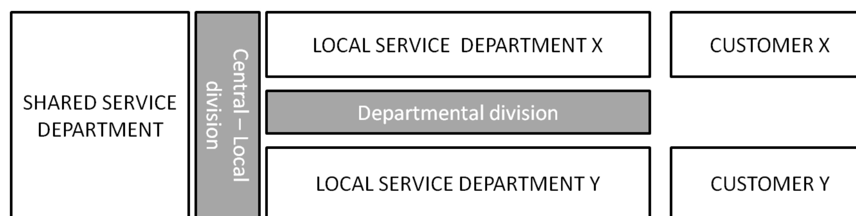

Fig. 1. The 'multi-' company IT department

On the left side of the figure we see the shared service department (or central IT department) which coordinates the whole IT structure and has an overview over all the IT projects in the different local service departments (or local IT departments). These local departments each provide software for their specific customer (or group of customers). Depending on the type of 'multi-' company, the customer can be a business line, a business unit, a market segment,… Each local department is thus coupled with a customer(group). The task of the local departments is to provide customized software systems to the customer they have been assigned to. The division between the central and the local departments on the one side, and the division between the local departments on the other side are both necessary in order to ensure the scalability of the department. If the different tasks for the different parts of the IT department are not clearly defined, it will become difficult to increase the size of the IT department.

Although other organizational blueprints are definitely possible, we opt for this construct because it is used by both existing companies and suggested by consulting firms. The most important aspects of the blueprint are the actors and their responsibility. In the next section we will elaborate our framework based on the organizational actors presented here.

5 Variability in software product line requirements

The second part of our framework concerns the variability of software product line requirements. As mentioned before, variability is a central concept in SPLE and therefore central to our starting framework. Reasoning *explicitly* about variability is something that really needs to be done if you want a SPLE project to have any chance on success. The earlier in the software engineering lifecycle decisions about variability are made explicit, the more effect they will have. Ignoring variability altogether makes it likely that implicitly some decisions concerning variability are already made. These decisions may not always be optimal, the risk of taking suboptimal decisions increases if variability is not explicitly dealt with. Besides the fact that wrong decisions will be made and extra costs will incur to correct these mistaken decisions, the total cost amount will also rise if the mistakes are only corrected later on in the process. Correcting the architecture of a domain asset before it is created is much cheaper than correcting all the implementations that are already built upon a faulty domain asset separately.

Our methodology will support making decisions about variability explicitly and as early as possible. During the requirements negotiation with the different stakeholders of a product line, it is easy to explicitly interact about variability. When multiple stakeholders have different demands and requirements for the information systems to be developed, these differences can be identified and reasoned upon. Reasoning about variability during requirements engineering is split up into two phases.

The first phase is the phase during which all stakeholders define their requirements and based on these requirements it is decided which requirements are common for all stakeholders (the commonality of the requirements) and which ones are different (the variability of the requirements). During this decision process differences between requirements can be discussed, in an attempt to slightly alter the requirements so they will possess more commonality and less variability. The drive for this harmonization step is the fact that the commonality part can be reused without making extra costs.

During the second phase it is decided which parts of the variable requirements will be implemented in the domain asset as variability (that is supported by the domain asset) and which variable requirements are left to the local service departments to implement themselves on a customer by customer basis. Supported variable requirements are those requirements that the central (shared service) department agrees to support. The shared service department implements these variable requirements as variation points [8] in the architecture of the domain assets. These variation points can be filled in by any of the variants that are offered by the central department. Therefore it can be said that the variable requirements that are supported in the domain asset are available for every implementation, and that that part of the variability is shared by all implementations. The other part of the variable requirements is specific to some stakeholders and therefore these are not supported centrally. The reason for supporting some variability centrally is once again for cost reduction purposes. The second phase is called variabilization, because this part of the decision process concerns defining up to what point variability is supported by the shared service department.

The following figure (Fig. 2) visualizes the variability decision process during requirements engineering for a product line. Each phase is visualized by its respective arrow next to the octagonal, which represents the total collection of requirements. On the left, with arrow number 1, we have the harmonization phase which decides on the amount of commonality and variability in the requirements. On the right we have the variabilization phase, labeled number 2. It is during this phase we decide which part of the variable requirements is shared in the form of variation points in the domain asset, and which part of the variable requirements is specific for certain stakeholders and left to be implemented by the local service departments.

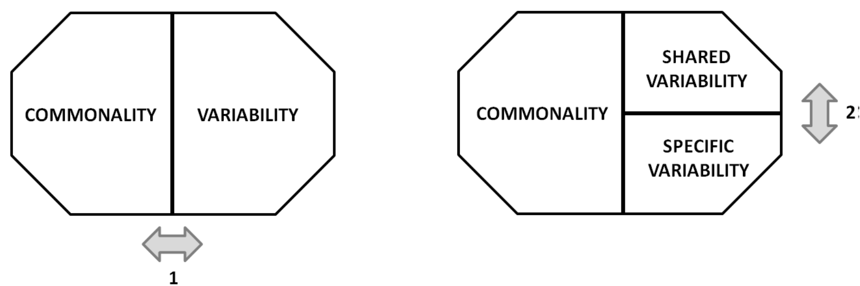

Fig. 2. Requirements harmonization (1) and variabilization (2)

6 Variability in model specifications

In the previous section we made explicit decisions about variability, but the results of these decisions need to be written down. Different modelling techniques available in literature describe the requirements of a software product line. These modelling techniques all represent variability in their own way. The problem however is linking these different kinds of representations correctly so that they represent the same variability. Common to the techniques is the fact that they represent some kind of decomposition possibility and that the variability is incorporated in it. We focus on feature and goal models, as they are the most popular requirements modelling techniques in early requirements engineering.

Probably the best known product line modelling technique is the feature model [12]. A feature model visualizes all possible feature combinations for one product line. A feature is a logical grouping of requirements which represent a certain aspect of a software system. The visualization of the features is done by a feature diagram [12]. This is an acyclic graph, and most of the times even a hierarchical tree. Features can be classified into groups in different ways. A first classification can be done based on the functionality which is represented. A feature can represent a group of requirements which are either functional or non-functional [13]. Functional requirements and features demand the development of some functionality of the information system. Non-functional requirements and features represent demands that link to certain performance or service levels, but do not demand any functionality.

The second manner of feature classification is more interesting in the context of

variability and is based on the necessity of the feature in its context of the software product line. There are three different categories of features in a feature model based on their necessity: mandatory features, optional features and alternative features. These different kinds of features all have a distinct visualization in the feature diagrams so that they are easy to identify. The optional and the alternative features are the ones that represent the variability in a software product line, and the link with variability is exactly what we are looking for.

The second technique which is gaining more and more interest in current research on requirements engineering is goal modelling. According to van Lamsweerde [14], a goal is a prescriptive statement of intent that the system should satisfy through the cooperation of its agents. A goal is thus something that needs to be attained by certain system components in order to be fulfilled. A system component can be a human actor with a specific role, a software component, a measuring device (e.g. a sensor),… Variability in goal models is represented by alternative refinements.

The difference between a requirement and a goal is the following: a requirement is a prescriptive statement that needs to be enforced by the software system being developed, a goal is a prescriptive statement that can be enforced by any system component. Goals are therefore useful in a context where there are business stakeholders (or other non IT stakeholders) present, while requirements present the possibility for IT stakeholders to focus on the software system which needs to be developed. In a SPLE project both modelling techniques have their advantages, so both should be used. When using both modelling techniques, one should be careful that both techniques represent the same variability. Research about linking goal models and feature models is thus needed in order to ensure that when variability is explicitly dealt with, it is being represented correctly. Even during negotiations before decisions are taken, it is beneficial to link feature and goal models, to facilitate the negotiation process, hence the inclusion in our initial framework.

7 Software product line engineering with explicit variability

The last theoretical pillar of the initialframework concerns the total engineering process of a software product line. In the previous section we stressed the importance of requirements engineering, but all other phases in the engineering process need to cope with the identified variability. Typically SPLE is divided into two process cycles, namely domain engineering and application engineering [15]. The domain engineering cycle is concerned with the development of domain assets. Once the domain assets are developed, these assets can then be reused in the application engineering cycle each time an instance of the product family needs to be developed.
The VeSPREM framework slightly alters the traditional 2 cycle development by extending it with one extra cycle. The first cycle is the same as the domain engineering cycle, during which the assets are developed with special attention devoted to variability. The second cycle is the application instantiation cycle. This cycle roughly matches the traditional application engineering cycle. The variability points in the domain asset are bound with specific variants and the asset is then localized (or implemented specifically) for the customer in question. The third and last cycle is an

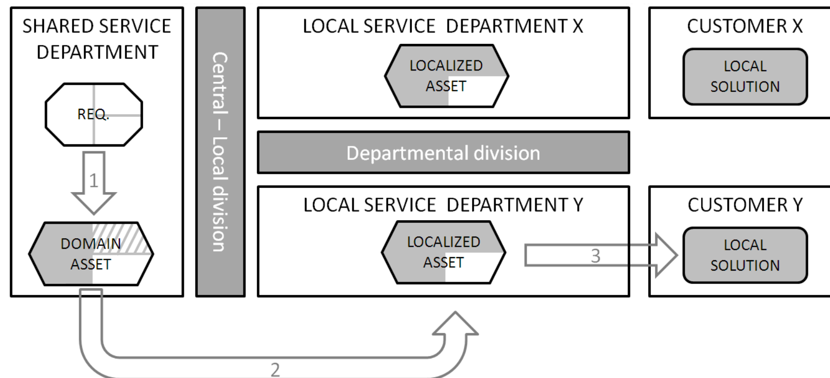

Fig. 3. Product line engineering cycles

additional cycle not included in the traditional bi-cycle process which deals with the customer specific extension of the now localized asset by the local departments. In traditional SPLE this last cycle is (implicitly) part of the application engineering cycle, but we suggest making this distinction because the extending activity clearly differs from the localizing activity. Localizing is done in the same way with the same set of possible variants for all product line members, while extensions are de facto member specific. Because different rules and forces can apply in both phases they should be split up instead of being considered as one application engineering step.
In Fig. 3 the three cycles are represented by numbered arrows. Number 1 is the domain asset cycle where the common part and the supported variability part are created, number 2 is the localizing cycle where the supported variability is filled in depending of the context and number 3 represents the extension cycle where local additions are made. The cycles are mapped against the IT department structure of section 4 in order to show the natural mapping of it on the cycles.

8 VeSPREM: a conclusion and look at the future

In the previous sections we have described each pillar of our framework. As presented in this paper, the framework only serves as a starting point for the case studies we plan to conduct and by no means yet fixed nor claimed to be complete. One possible extension of the framework which will be studied during the case studies is the possibility to create a formal method for transformations between goal models and requirement models. Another addition to the framework currently under study is the extension from traditional SPLE towards Service Oriented Product Line Engineering (SOPLE). Further extensions will also definitely arise when we are coupling the case study results back to the framework. The longer term objective of our research is to enhance and fill the framework to such a level we obtain a solid and complete methodology concerning variability inside software product lines.

As made clear by the explanation of the pillars of the framework, the theory for successful SPLE is available in research, but the link towards practice still lacks

strength, and this is where VeSPREM will ultimately prove useful. Once the methodology is completed, the following step will be to introduce VeSPREM into companies and in that way create new opportunities for further explorations in empirical research.

Acknowledgements

We gratefully acknowledge KBC Global Services NV (member of the KBC Group) for funding this research through the KBC research chair "Developing and Managing Business Services as Shared Assets".

References

1. Kang, K.C., Cohen, S.G., Hess, J.A., Novak, W.E., Peterson, A.S.: Feature-Oriented Domain Analysis (FODA): feasibility study. Software Engineering Institute, Carnegie Mellon University (1990)
2. Clements, P., Northrop, L.: Software product lines: practices and patterns. Addison-Wesley Longman Publishing, Boston (2001)
3. Alves, V., Niu, N., Alves, C., Valença, G.: Requirements engineering for software product lines: A systematic literature review. Information and Software Technology 52, pp. 806 -- 820 (2010)
4. Hubaux, A., Heymans, P., Benavides, D.: Variability Modelling Challenges from the Trenches of an Open Source Product Line Re-Engineering Project. In International Conference on Software Product Lines, pp. 55 -- 64, IEEE Computer Society, Limerick (2008)
5. Linden, F. J., Schmid K., Rommes, E.: Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering. Springer-Verlag, Berlin (2007)
6. Lee, K., Kang, K.C., Lee, J.: Concepts and Guidelines of Feature Modeling for Product Line Software Engineering. In International Conference on Software Reuse, pp. 62 – 77, Springer-Verlag, Austin (2002)
7. Chen, L., Ali Babar, M., Ali, N.: Variability management in software product lines: a systematic review. In: Proceedings of the 13th international Software Product Line Conference, pp. 81 -- 90, Carnegie Mellon University, San Francisco (2009)
8. Kim, S.D., Her, J.S., Chang, S.H.: A Theoretical Foundation of Variability in Component-Based Development. Information and Software Technology. 47, pp. 663 -- 673 (2005)
9. Svahnberg, M., van Gurp, J., Bosch, J.: A Taxonomy of Variability Realization Techniques. Software Practice & Experience 35, pp. 705 – 754 (2005)
10. Travers, M.: Qualitative research through case studies. Sage Publications Inc., Los Angeles (2001)
11. Yin, R.K.: Case Study Research: Design and Methods. Sage Publications Inc., Los Angeles (2009)
12. Czarnecki, K., Eisenecker, U. W.: Generative Programming: Methods, Tools, and Applications. ACM Press/Addison-Wesley Publishing Co., New York (2000)
13. Robertson, S.,Robertson, J.: Mastering the Requirements Process. ACM Press/Addison-Wesley Publishing Co., New York (1999)
14. van Lamsweerde A.: Requirements Engineering: From System Goals to UML Models to Software Specifications. Wiley, Chichester (2009)
15. Pohl, K., Böckle, G., Linden, F. J.: Software Product Line Engineering: Foundations, Principles and Techniques. Springer-Verlag, New York (2005)