Paper II

*The PLUSS Approach - Domain Modeling with Features, Use Cases and Use Case Realizations

Magnus Eriksson[1], Jürgen Börstler[2] and Kjell Borg[1]

[1] Land Systems Hägglunds AB, SE-891 82 Örnsköldsvik, Sweden
{Magnus.Eriksson, Kjell.Borg}@baesystems.se
[2] Dept. of Computing Science, Umeå University, SE-901 87 Umeå, Sweden
jubo@cs.umu.se

Abstract. This paper describes a product line use case modeling approach tailored towards organizations developing and maintaining extremely long lived software intensive systems. We refer to the approach as the PLUSS approach, *Product Line Use case modeling for Systems and Software engineering*. An industrial case study is presented where PLUSS is applied and evaluated in the target domain. Based on the case study data we draw the conclusion that PLUSS performs better than modeling according to the styles and guidelines specified by the IBM-Rational Unified Process (RUP) in the current industrial context.

* Proceedings of of the 9'th International Conference on Software Product Line, LNCS, Vol. 3714, Springer-Verlag, pp. 33-44

1 Introduction

Software intensive defense systems, for example vehicles, are developed in short series. They are always customized for different customer needs and they are expected to have an extremely long life span, often 30 years or longer. For an organization to be competitive in a market like this it is important to achieve high levels of reuse and effective maintenance. An interesting approach to address issues like these, which has gained considerable attention both by industry and academia over the last few years, is known as software product line development. The basic idea of this approach is to use domain knowledge to identify common parts within a family of products and to separate them from the differences between the products. The commonalties are then used to create a product platform that can be used as a common baseline for all products within the product family.

For embedded software we believe it is important that product line concepts such as domain modeling are also introduced into the systems engineering process, since embedded software requirements are for the most part not posed by customers or end users, but by systems engineering and the systems architecture. Due to earlier positive single system experiences with use cases, we are therefore interested in identifying a use case driven product line approach that can be applied by both our systems and software engineering teams. Unfortunately, we see a number of problems with existing approaches to product line use case modeling. To address issues in existing approaches we have developed a domain modeling approach that utilizes features [10], use cases and use case realizations [12]. For the remainder of this paper the approach will be referred to as PLUSS (*Product Line Use case modeling for Systems and Software engineering*).

The UML Use case meta-model [19] provides poor assistance in modeling variability [16]. A number of suggestions addressing this issue are described in the literature. Von der Maβen and Lichter suggest that the UML use case meta-model should be extended by two new relationships, “Option” and “Alternative” [16]. Jacobson et al. suggest using the “generalization” and “extend” relationships to model variability in UML use case diagrams [9]. We do however see a fundamental problem with using use case diagrams for describing variants. Use case diagrams tend to get cluttered to a degree where it is impossible to get an overview of the variants within a family. It is furthermore not enough to only manage variability among whole use cases. It must also be possible to specify variant behavior within use cases. There have been some proposals on how to do this in the literature, for example the PLUC notation [5] and RSEB parameters [9]. However, like the UML approaches above these approaches do not have any means to provide a good overview of the variants within a family. Most existing product line use case modeling approaches also lack strong mechanisms to trace variant behavior to the system design and they are document, not model driven. Using documents instead of a common model is a major maintenance concern working on extremely long lived systems. Product instantiation in a document driven approach typically involves copying documents and removing variant information. This is not good from a long term maintenance perspective since information is being duplicated.

Our approach is based on the work by Griss et al. on FeatuRSEB [8]. Like Griss et al. we argue that feature models are better suited for domain modeling than UML use case diagrams and that a feature model therefore should be used as the high level view of a product family. In FeatuRSEB a feature model is added to the 4+1 view model adopted by Jacobson et al. in RSEB [9]. The feature model in FeatuRSEB takes "center stage" and provides a high-level view of the domain architecture and the reusable assets in the product family. Even though a feature model is also used in our approach to provide a high-level view of the variability within a product family, a fundamental difference exists between PLUSS and FeatuRSEB. In PLUSS the primary purpose of the feature model is not to take "center stage", but rather to be a tool for visualizing variants in our abstract product family use case model. We maintain one complete use case model for the whole system family and we use the feature model as a tool for instantiating that abstract family model into concrete product use case models for each system built within the family.

The main contributions of this paper are: An improved approach to manage variant behavior in use case models, stronger means to trace variant use case behavior to the system design and stronger means to generate product use case models from a common family model.

The remainder of the paper is organized as follows: Section 2 provides an introduction to PLUSS feature modeling. Section 3 describes PLUSS Use case modeling and how the PLUSS feature model relate to the use cases. Section 3 also describes the PLUSS notation for describing variants in use case scenarios and how product use case models are instantiated form a family model. Section 4 presents an industrial case study in which the PLUSS approach is applied and evaluated in its target domain. In section 5, we summarizes the paper and draw conclusions.

2 Feature Modeling

Kang et al. first proposed use of feature models in 1990 as part of the Feature Oriented Domain Analysis (FODA) [10]. A feature is defined as a prominent or distinctive user-visible aspect, quality or characteristic of a system in FODA. In feature models, features are organized into trees of AND and OR nodes that represent the commonalties and variations in the modeled domain. General features are located at the top of the tree and more refined features are located below. Originally, FODA described "Mandatory", "Optional" and "Alternative" features that may have the relations "requires" and "excludes" to other features. Mandatory features are available in all systems built within a family. Optional features represent variability within a family that may or may not be included in products. Alternative features represent an "exactly-one-out-of-many" selection that has to be made among a set of features. A "requires" relationship indicates that a feature depends on some other feature to make sense in a system. An "excludes" relationship between two features indicates that both features can not be included in the same system.

FODA has no defined mechanism to specify the relation "at-least-one-out-of-many" [6]. Our experience has shown that this is an important shortcoming. We address this issue by defining a new feature type called "Multiple Adaptor" in

PLUSS. This feature type is similar to FODA's alternative features, but instead of representing the "exactly-one-out-of-many" relationship, it captures the missing relationship. Its name follows the naming scheme proposed by Mannion et al. for the equivalent relation in their work on reusable requirements [14]. We have also chosen to rename alternative features to "Single Adaptor" features following the same naming scheme. The feature modeling notation used in PLUSS is based on the FODA notation but it has been slightly modified to better suit our modeling needs as shown in Fig. 1. A filled black circle represents a mandatory feature and, as in the original notation, a non-filled circle represents an optional feature. Single and multiple adaptor features are represented by the letters 'S' and 'M' surrounded by a circle.

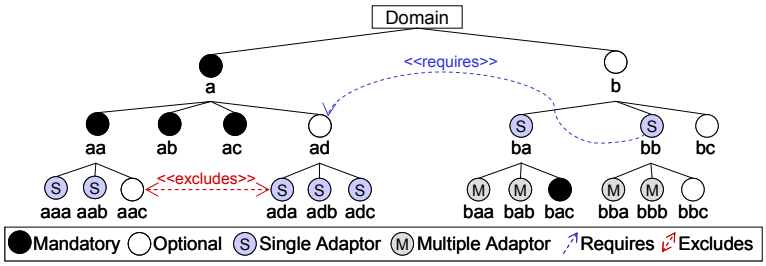


Fig. 1: An example feature model in the PLUSS notation.

To further clarify the PLUSS notation, we have created a mapping between PLUSS feature constructs and multiplicities [19] as shown in Fig. 2. As shown in Fig. 2 we have also identified a feature construct that should be avoided. Our experience has shown that this construct, a set containing only optional feature leaf nodes, encourages misuse of the refinement relation used for building the feature tree. This construct typically appear when a set of multiple adaptor features is mistaken for a set of optional features.

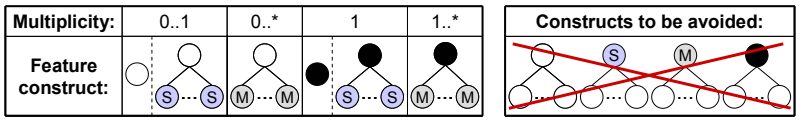


Fig. 2: Feature constructs vs. multiplicities, and constructs to be avoided in PLUSS.

One shortcoming of the PLUSS feature modeling notation, compared to for example Czarnecki et al. more expressive Cardinality-based notation [2], is the inability to model n..m multiplicity. Our experience has however shown that such constructs are not needed to capture the different types of variability the can exist in product family use case models. We therefore exclude cardinalities from our notation for the purpose of improved readability.

3 Use Case Modeling

As we described in [4], we have chosen to adopt the so called "Black Box Flow of Events" notation described in the Rational Unified Process for Systems Engineering

(RUP-SE) [17] shown in Fig. 3(a) for describing use case scenarios. This notation is used for tabular descriptions of use case scenarios in natural language. We argue that the notation has two major advantages over tradition natural language scenario descriptions. It forces analysts to always think about interfaces since separate fields exist for describing actor and system actions. It also provides a strong mechanism to relate non-functional requirements to use cases using the "*Blackbox Budgeted Requirements"* column.

A use case realization describes how a particular use case is realized within the system design in terms of collaborating design elements [12]. As we described in [4], we have chosen to describe use case realizations in natural language description based on the RUP-SE "White Box Flow of Events" [17] shown in Fig. 3(b). We have chosen natural language descriptions of use case scenarios and use case realizations since the PLUSS approach must be applicable for both systems and software engineering. This increases the number and diversity of stakeholders interested in the models and thereby makes for example UML unsuitable for the purpose. Our natural language descriptions can however be supplemented with UML diagrams as needed.

Step	Actor Action	Blackbox System Response	Blackbox Budgeted Req.
1	The **Actor**…	The **System**…	It shall…
2	…	…	…
3	The use case ends when…	…	…

(a)

Whitebox Action	Whitebox Budgeted Req.
DesignElement_1…	It shall…
DesignElement_2…	…
DesignElement_3…	…
…	…
…	…
…	…
…	…
…	…
…	…

(b)

Fig. 3: The (a) Blackbox flow of events used for describing use case scenarios, and (b) the Whitebox flow of events used for describing use case realizations.

3.1 The PLUSS Approach to Modeling Variants in Use Case Models

As mentioned in section 1, the basic idea of PLUSS is to maintain one common and complete use case model for whole product family. To do this, it must be possible to manage variability in the model. We have identified four types of variants that can exist in use case models for product families. The first type regards whole use case that can vary between systems built within a product family. We model this by relating one or more use cases with a feature of any type in the feature model. The second type of variability regards the set of included use case scenarios within each use case. We model this by relating one or more scenarios with a feature of any type in the feature model. The third type regards the set of included steps in each use case scenario. We model this by relating scenario steps with features of any type in the feature model. The fourth and final type of variability regards cross-cutting aspects that can affect several use cases on several levels. Cross-cutting aspects are modeled as use case parameters in PLUSS, these parameters must be related to a set of single

adaptor features in the feature model. Gomaa [7] proposed to model each feature as a use case package. PLUSS extended this idea, saying that possibly a whole set of features compose a use case package. This have the advantage of enabling us to also visualize variants within use cases specifications using the feature model.

A meta-model for integration of features, use cases and use case realizations is shown in Fig. 4**.** It describes how use cases, scenarios and scenario steps are included by feature selections. This meta-model is an extension of the meta-model presented in [4] that also show how these included use case scenario steps prescribes a certain set of design element via use case realizations. Variant use case behavior is thereby traced to the system design.

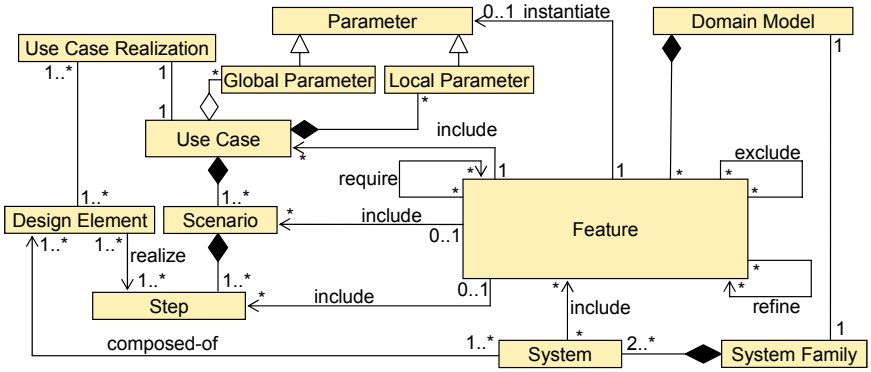


Fig. 4: The PLUSS Meta-model.

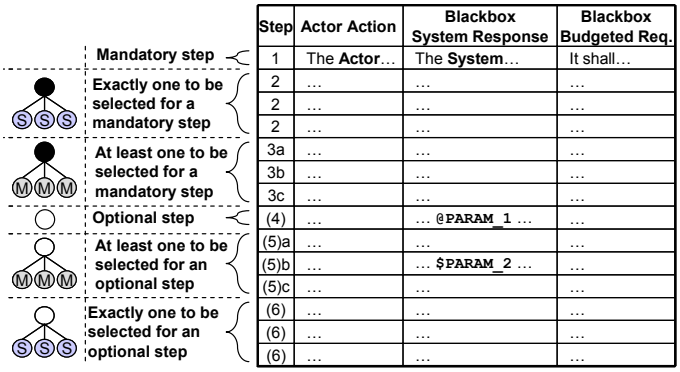
Change cases, first proposed by Ecklund et al. [3], are basically use case that specifies anticipated changes to a system. Change cases also provide the relation "impact link" that creates traceability to use cases whose implementations are affected if the change case is implemented. In PLUSS, change cases are primarily used to mark proposed, but not yet accepted functionality in a domain. New requirements are first modeled as change cases, however once accepted for implementation in a system within a family, these change cases are transformed to use cases.

3.2 The PLUSS Notation for Describing Variants in Use Case Specifications

As we described in [4], the step identifier of the blackbox flow of events notation discussed in section 3 can be extended to describe variants in use case scenarios as shown in Fig. 5. A step identified by a number describes a mandatory step in the scenario, as it does in the original notation. Several steps identified with the same number identify a number of mutually exclusive alternatives for one mandatory step in the scenario. These steps must be related to a set of single adaptor features with a mandatory parent in the feature model. Several steps identified with the same number and a consecutive letter identify a number of alternatives for one mandatory step in the scenario out of which at least one must be selected. These steps must be related to a set of multiple adaptor features with a mandatory parent in the feature model. A step identified by a number within parenthesis identifies an optional step in the scenario. Optional steps must be related to an optional feature in the feature model. Several

steps identified with the same number within parenthesis and a consecutive letter identify a number of alternatives for one optional step in the scenario out of which at least one must be selected. These steps must be related to a set of multiple adaptor features with an optional parent feature in the feature model. Several steps identified with the same number within parenthesis identify a number of mutually exclusive alternatives for one optional step in the scenario. These steps must be related to a set of single adaptor features with an optional parent in the feature model.

Jacobson et al. introduced the concept of use case parameters as part of the RSEB in [9]. Mannion et al. distinguished between local parameters and global parameters in their work on reusable natural language requirements [14]. We find this distinction useful also when working with use cases. In PLUSS, the scope of a local parameter is the use case in which it resides and the scope of a global parameter is the whole domain model. Like Mannion et al. we use the symbols '$' and '@' respectively to denote local and global parameters as shown in step '(4)' and '(5)b' of Fig. 5.



Step	Actor Action	Blackbox System Response	Blackbox Budgeted Req.
1	The **Actor**…	The **System**…	It shall…
2	…	…	…
2	…	…	…
2	…	…	…
3a	…	…	…
3b	…	…	…
3c	…	…	…
(4)	…	… `@PARAM_1` …	…
(5)a	…	…	…
(5)b	…	… `$PARAM_2` …	…
(5)c	…	…	…
(6)	…	…	…
(6)	…	…	…
(6)	…	…	…

Fig. 5: The PLUSS notation for describing variants in use case scenarios.

3.3 Product Instantiation in the PLUSS Approach

Although the actual organization may vary, typically, when a new product is going to be added to a product family, initial requirements analysis is performed by a product team. This analysis will result in a set of change requests (CR) regarding new requirements (change cases) to be added to the domain model and regarding features that should be included in the new system. The domain engineering team is then responsible for performing change impact analyses on the change requests. A domain engineering change control board (CCB) may then decide if the requested set of requirements will be allowed in the product. Since a common use case model is maintaining for a whole product family in PLUSS, product instantiation is then basically done by adding any new requirements to the model and then using the feature model to choose among its variants. The set of included features directly correspond to a specific set of included use cases for the product. A product use case model is then generated by applying a filter to the domain model sorting out features not included in the current system. This will result in three types of reports: A "*Use*

Case Model Survey" including all use cases included in the product, and "*Use Case Specifications",* and "*Use Case Realizations*" for all use case in the survey.

4 Case Study

The objective of this case study was to apply the PLUSS approach in the target domain to evaluate its feasibility. The hypothesis to be tested in the method evaluation and its null hypothesis were

H_1: *The PLUSS approach performs better than modeling according to the company process baseline in a product line setting.*

H_0: *The PLUSS approach performs equal to, or worse than the modeling according to company process baseline.*

A number of response variables relevant for measuring the performance of the approach were identified as part of the case study design. Examples are: *effort for learning and understanding notations* used; *effort for long term maintainability* of specifications; and *usefulness of the resulting models*.

4.4 Study Context

The case study was preformed with the Swedish defense contractor Land Systems Hägglunds. Land Systems Hägglunds is a leading manufacturer of combat vehicles, all terrain vehicles and a supplier of various turret systems. The company process baseline for software development, against which PLUSS was compared, is development according to the IBM-Rational Unified Process (RUP) [12].

The PLUSS approach was applied on the Vehicle Information System (VIS). The VIS subsystem is responsible for tasks such as displaying video, providing electronic manuals, performing onboard system test and diagnostics, displaying logs, displaying system status and reporting system alarms. The development of VIS has recently gone from clone-and-own reuse [1], to adopting a software product line approach. The transformation to software product line development was initiated by forming a domain engineering team which is now responsible for development and maintenance of the VIS core assets. At the time of the case study, the domain engineering team had successfully delivered core assets to their first customer project and was in the process of analyzing requirements for its second customer project.

The main CASE tools used for supporting the PLUSS approach were the requirements management tool Telelogic DOORS and the UML modeling tool IBM-Rational Rose. Rose was used for drawing feature graphs and UML Use case diagrams. DOORS was used for managing the overall domain model. Each feature was represented as an object in the database with a number of attributes; like feature type, products including the feature and a use case diagram. Each use case was represented as a module in DOORS. Scenario steps, both blackbox and whitebox, were represented as objects in those modules. Traceability links were used to relate features to use cases, scenarios and scenario steps according to the PLUSS meta-

model shown in Fig. 4. A small number of scripts were written in DOORS to aid the modeling.

The domain modeling activity stared with a four hour introductory lecture on the PLUSS approach to the domain engineering team. After the lecture, the domain team had a four hour brainstorming session identifying and documenting features in the feature model. After this session, the domain engineering team split-up and only the product line analysis team continued the domain modeling for the reminder of the study. The product line analysis team consisted of three people, out of which two performed most of the modeling activities and the third mainly acted as a tool specialist, responsible for customizing DOORS to better support PLUSS.

4.5 Method

The case study involved collecting data from four different types of sources. The first type of data was collected by *examining documentation* [18]. Modeling artifacts from the early phases of the project were inspected to verify that they where used in the proper manner. The second type of data was collected by *participant observation* [18]. The research team assumed a mentoring role for the product line analysis team and could thereby get first hand information about any problems they ran into during the modeling activities. The third type of data was collected through *questionnaires* [11]. During the evaluation the product line analysis team filled out a questionnaire describing their experiences applying the approach. The questionnaire was designed to have both specific and open ended questions to also elicit unexpected types of information. The final type of data was collected trough *interviews* [18]. A total number of nine people, representing the domain engineering team, the product development team, the systems engineering team and technical management were interviewed to gather their views on the usefulness of the models and on possible pros and cons with the PLUSS approach. Interviews began with a short introduction to the research being performed. After the introduction, the VIS domain model and a product instance of the model were shown and discussed with each interviewee. Interviews proceeded in a semi-structured manner, trying to elicit as much information as possible about opinions and impressions regarding PLUSS.

The different types of data collected were first analyzed individually to find patterns and trends in the responses, then analyzed all together and conclusions were drawn about the case study hypothesis.

4.6 Threats to Validity

To minimize threats to the study's *construct validity,* the case study hypothesis and its null hypothesis were stated as clearly and as early as possible in the case study design to aid in identifying correct and relevant measures [11]. To minimize threats to the study's *internal validity,* the case study project was staffed using the organizations normal staff-allocation procedures. Everyone involved in the case study had good knowledge of modeling according to the company process baseline, against which the PLUSS approach was compared [11]. Furthermore, interviewees were chosen in

collaboration with the organization's management to ensure that they properly represented their group of stakeholders. To avoid *Howthorne effect* [15], attitudes towards the company process baseline were collected from subjects and taken into account during data analysis. It was also pointed out to subjects that no "correct" answers existed, and that it was important that their answers correctly reflected their view. One confounding factor that may have affected the internal validity of the study is the close involvement of the research team with the product line analysis team. We do however judge this risk to be minor since the domain analysis team performed the actual modeling themselves and the mentoring activity mainly consisted of discussion meetings where possible problems were raised and discussed. To minimize threats to the study's *external validity,* the case study was conducted in the target domain of extremely long-lived software intensive systems and the pilot project was selected to be of typical size and complexity for the organization [11]. To minimize threats to the study's *conclusion validity,* results were triangulated by collecting data with four different methods from several different sources. Furthermore, results were discussed with the teams to assure that their opinions were represented correctly [18].

4.7 Results

Document examination indicated that the team understood and was able to apply all notations used after only the four hour introduction to the approach, even tough they had no earlier experience of feature modeling.

Participant observation revealed two initial problems applying PLUSS. During the first brainstorming session, the domain engineering team misused the feature model to "invent" variability that would force a "beautiful implementation", instead of focusing on creating a reusable requirements model. This problem was however resolved when the issue was discussed at the first mentoring meeting. The second problem regarded maintaining correct abstraction level. Even tough the team was to model only a certain subsystem (VIS), sometimes also system level functions appeared in the models. This problem was however resolved when the research team introduced a system context diagram [13] in the modeling process.

Questionnaires indicated that the product line analysis team gained a better understanding of the domain during the modeling activity. The team felt that applying PLUSS was an overall positive experience and that PLUSS has a number of positive characteristics, for example its way of providing a total overview of the product family and the possibility to maintain a common model for a whole family. A problem pointed at in the open ended questions was that the domain analysis team felt that DOORS and Rose were not integrated well enough, and that this resulted in time consuming manual synchronization of the models. However, as shown in Fig. 6, questionnaires indicated that the PLUSS approach performed better than the company baseline in the VIS context.

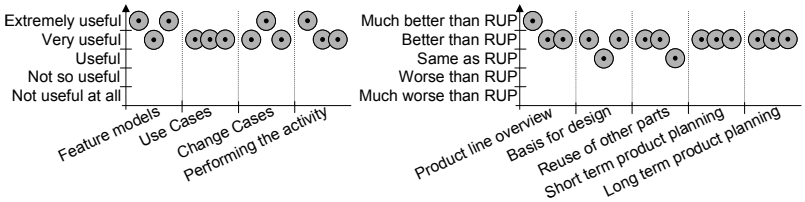


Fig. 6: Overview of questionnaire results, (a) usefulness of concepts / performing the modeling and (b) usefulness of resulting models compared to the company baseline.

Interviews with *product line analysts* indicted that the PLUSS approach provides a better overview of the product line. The team also believed that the approach will improve the overall quality of the models and ease their maintenance. Experience of clone-and-own reuse [1] of use cases in earlier projects had pointed out a maintenance problem which they believed PLUSS addresses. They could not identify any scalability problems with the approach. However, they did believe that for it to work well, smart decisions from technical management regarding scooping and a strong configuration management function is needed. Analysts believed the initial extra investment related to applying the PLUSS approach would be returned in terms of reduced modeling costs already in the second or third project applying the approach.

Interviews with *product line designers* indicted that notations used were easy to understand and that the resulting models provided a good overview of dependencies within the model. They also felt that the approach made models more coherent and easier to find information in. They believed that the PLUSS approach will significantly increase the quality of specifications and ease their maintenance. Designers felt that change cases "might be good to keep in mind", but a "probability of implementation" attribute would increase their usefulness. Designers could not identify any scalability problems with the approach. However, they did believe it to be important that technical management try to keep the number of variants down.

Interviews with the *product development team* indicated that the PLUSS approach offered product line mechanisms significantly stronger than anything the RUP has to offer. They believed that PLUSS will significantly reduce the effort needed for requirements analysis and that it has potential to largely reduce the amount of specification work. The team could not identify any scalability problems with the approach. They did however see a risk that the number of features might explode if too much new functionality is added in each project. They therefore believed a strong management function is needed keep the number of variants down. They also identified a risk that adding one or a few new features might create a dependency explosion in the feature graph, since the model is closely related to business rules. This thought could however not be further elaborated or illustrated by the team. The team also identified a need for obsolete management of features to prevent the feature tree from growing to infinity. The product development team believed the initial extra investment related to applying the PLUSS approach would be returned in terms of reduced modeling costs already in the second project applying the approach.

Interviews with the *systems engineering team* indicated that the notations used were easy to understand also for personnel with a non-software background. They liked the idea of a common model being a central source of information about a

domain. They also found the use of change cases to tag unimplemented functionally very useful since it provides a good overview of what is new and what has been done before. They believed that the resulting models would be a good tool for early cost estimates and that the approach would encourage and produce high levels of reuse. The systems engineering team could not identify any problems with PLUSS. They did however see a risk with the whole concept domain modeling and requirements reuse. They believed that it might cause an organization to loose its visions and thereby cause products to stop evolving. Systems engineering also expressed a need for stronger means to document design rationale. This was however not seen as a problem with PLUSS, but as an important supplement to be further investigated.

Interviews with *Technical Management* indicate that the PLUSS approach provides significantly stronger support for product planning than traditional RUP. Management liked the fact that it is a use case driven approach, and the idea of a central source of information about a domain. Management also felt that feature models provided a good overview of the requirements space for the domain and that change cases provided a good overview of the current delta. However, to further improve the utility of change cases, management would like change cases to have attributes specifying planed platform release supporting them. Management also believed that PLUSS models could be a powerful means of communication towards other parts of the organization. Management believed the initial extra investment related to applying the PLUSS approach would be returned in terms of reduced modeling costs already in the second project, assuming the domain engineering team was able to produce models of required quality before the start of the second project.

5 Summary and Conclusions

We have described how a common use case model can be developed and maintained for a whole family of products in PLUSS. We have also described how product use case models can be generated from a family model by selecting features from a feature model. The approach was applied and evaluated in an industrial case study in the target domain. Triangulating on the collected case study data has led us to reject the case study null hypothesis. We thereby draw conclusion that the PLUSS approach performs better than modeling according to the styles and guidelines specified by the RUP in the current industrial context. Results did however also indicate that for PLUSS to be successfully applied, stronger configuration management and product planning functions than traditionally found in RUP projects are needed. Furthermore, results also pointed at a need for better tool support and stronger means to document design rationale. We consider these areas to be important areas of future work.

References

1. Bosch, J.: Design & Use of Software Architectures, Addison-Wesley (2000)

2. Czarnecki K., Helsen S., Eisenecker U.: Staged Configuration Using Feature Models, Proceedings of the Software Product Line Conference (SPLC 2004), LNCS 3154, Springer-Verlag, (2004) 266-283.
3. Ecklund E., Delcambre L., Freiling M.: Change Cases - Use Cases that Identify Future Requirements, Proceedings of OOPSLA'96, San Jose, Ca, October 6-10, (1996) 342-358.
4. Eriksson M., Börstler J., Borg K.: Marrying Features and Use Case for Product Line Requirements Modeling of Embedded Systems, Proceedings of the Fourth Conference on Software Engineering Research and Practice in Sweden SERPS'04, Institute of Technology, UniTryck, Linköping University, Sweden (2004) 73-82
5. Fantechi A., Gnesi S., Lambi G., Nesti E.: A Methodology for the Derivation and Verification of Use Cases for Product Lines, Proceedings of the International Conference on Software Product Lines, Lecture Notes in Computer Science, Vol. 3154, Springer-Verlag (2004) 255-265
6. Fey D., Fajta R., Boros A.: Feature Modeling: A Meta-model to enhance Usability and Usefulness, Proceedings of the International Conference on Software Product Lines, Lecture Notes in Computer Science, Vol. 2371, Springer-Verlag, (2002) 198-216.
7. Gomaa H.: Designing Software Product Lines with UML – From Use Cases to Pattern-Based Software Architectures, Addison-Wesley (2004)
8. Griss M., Favaro J., d'Alessandro M.: Integrating Feature Modeling with the RSEB, Proceedings of the Fifth International Conference on Software Reuse, Vancouver, BC, Canada, (1998) 76-85.
9. Jacobson I., Griss M., Jonsson P.: Software Reuse – Architecture, Process and Organization for Business success, Addison-Wesley (1997)
10. Kang K. Cohen S., Hess J., Novak W., Peterson A.: Feature Oriented Domain Analysis (FODA) Feasibility Study, Technical Report CMU/SEI-90-TR-021, Software Engineering Institue, Carnegie Mellon University, Pittsburgh, PA (1990)
11. Kitchenham B., Pickard L., Pfleeger S.: Case Studies for Method and Tool Evaluation, IEEE Software, Vol. 12 Nr. 45 (1995) 52-62
12. Kruchten P.: The Rational Unified Process - An Introduction, Second Edition, Addison-Wesley (2000)
13. Lykins H., Friedenthal S, Meilich A.: Adapting UML for an Object Oriented Sysyems Engineering Method (OOSEM), Proceedings of the 10'Th International INCOSE Symposium (2000)
14. Mannion M., Lewis O., Kaindl H., Montroni G., Wheadon J.: Representing Requirements on Generic Software in an Application Family Model, Proceedings of the International Conference on Software Reuse ICSR-6 (2000) 153-196.
15. Mayo E.: The human problems of an industrial civilization, New York: MacMillan (1933)
16. Von der Maβen T., Lichter H.: Modeling Variability by UML Use Case Diagrams, Proceedings of the International Workshop on Requirements Engineering for Product Lines (2002) 19-25
17. Rational Software: The Rational Unified Process for Systems Engineering Whitepaper, Ver. 1.1, Available at: http://www.rational.com/media/whitepapers/TP165.pdf, (2003)
18. Seaman C.: Qualitative Methods in Empirical Studies of Software Engineering, IEEE Transactions on Software Engineering, July/August (1999) 557-572
19. OMG: Unified Modeling Language Version 2.0, Available at: http://www.uml.org (2005)