Integrating Product Family Modeling with Development Management in Agile Methods

Mikko Raatikainen, Kristian Rautiainen, Varvana Myllärniemi, Tomi Männistö
Helsinki University of Technology
P.O. Box 9210 02015 TKK Finland
firstname.lastname@tkk.fi

ABSTRACT

Software product families and agile development have emerged as a popular means in software engineering. In this position paper, we discuss how development management in agile methods can be integrated with software product family structure modeling. The integration aims towards improving software product family development governance by providing technology in terms of concepts and even automated tool support for planning, monitoring and controlling the development work for different stakeholders. For example, integration provides support for prioritization of development tasks and enables monitoring the development status of products in a software product family, for example. The feasibility of integration is shown by combining Kumbang and Agilefant conceptualizations and respective prototype tools.

Categories and Subject Descriptors

D.2.9 [**Software Engineering**]: Management

General Terms

Design, management

Keywords

Backlog, software product family, modeling

1. INTRODUCTION

Today, many software product companies face a diversity of customer needs. Instead of developing a single product as a compromise of these needs, many companies offer several software products with slightly varying capabilities. For example, the products the companies develop typically need to be able to address multiple user segments, such as home and professional users; allow categorization from low- to high-end products; support various hardware platforms and operating systems; offer different sets of features for different needs; and cover cultural variability, such as legislation, language, and market structures in different countries. Software product family engineering has emerged as a means of developing such a set of products, the assets they consists of, and their structure, including, e.g., compositional structure, variability, dependencies, and rules within software product family [3, 8]. Another trend today is adopting agile development practices. While different agile methods have emerged, many of them share common characteristics such as iterative, incremental and time-paced development. In particular, software development work is typically managed using *backlogs* [10]. A backlog roughly refers to a prioritized list of things to get done. For example, a product backlog contains a prioritized list of items that need to be done in order to fulfill the product goals whereas an iteration backlog contains items to be developed within an iteration.

However, neither software development management using backlogs or product family engineering alone are sufficient to give full understanding of alignment between the product goals, current status of the development, and the product design. Rather, they provide relatively different views to software development. On the one hand, development management with backlogs provides a view to the product goals and how and when these goals are aimed to be realized, but does not take into account the actual design of the software product family. For example, a goal can be implemented by features that, however, require other features, which do not contribute directly to the goal. This dependency may go unnoticed during development planning which results in overhead due to re-planning when the situation is revealed or failing fulfillment of the goal. This kind of scenario is further aggravated if there are multiple agile teams developing features to the product family. On the other hand, software product family engineering focuses typically on structure and does not take into account when and how each artifact is developed and how long the development takes. For example, a structure of a software product family can be modeled but the model does not take into account when the system or a specific feature will be developed and finished. Consequently, it seems that it would be beneficial to integrate these two views. However, the relationship between product family engineering and development management using backlogs seems to be scarcely addressed.

In this position paper, we discuss integrating software product family modeling with development management by combining backlog management of agile development and structural modeling of software product families. The integration aims towards improving software product family

SDG'08, May 12, 2008, Leipzig, Germany.

development governance by providing concepts and even automated tool support for planning, monitoring and controlling the development work. For example, development planning and prioritization can be performed by analyzing the product structure so that all dependencies have been taken into account and, hence, a release is meaningful and consistent; or the items in a backlog provide estimates for effort and schedule, and current development status of a specific feature or a product configuration of the software product family, hence providing visibility into development progress. We limit the focus on enabling concepts and tools. Hence, despite being relevant, other issues such as organizational dynamics are left out of scope.

The rest of the paper is organized as follows. In Section 2, we outline software product family modeling and development management using backlogs as background of this work. In Section 3, we outline structural modeling in software product families and backlogs in term of two approaches called Kumbang and Agilefant, respectively. Further, we show feasibility of integration by combining the concepts in these approaches. Section 4 outlines an integrated tool support for backlog management and structural modeling on the basis of existing tool support of the approaches. Finally, Section 5 discusses implications such as benefits and applicability of the integration.

2. BACKGROUND

2.1 Software Product Family Engineering

A software product family is defined as "a set of software-intensive systems that share a common, managed set of features satisfying 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" [3]. The assets include, but are not limited to, software components or other development units. In particular, a special asset is a software product family architecture that defines the structure that the products must adhere to.

A special characteristic of the assets and architecture of a software product family is that they incorporate variability. *Variability* is the ability of a system to be efficiently extended, changed, customized or configured for use in a particular context [12]. By binding variability and potentially developing product specific code, the different products of a software product family can be derived.

Due to the importance of variability in a software product family, several modeling approaches have been studied. Feature models [6] are one of the first widely known approaches that take into account variability. A feature refers to a user visible characteristics of a system. Recently, other modeling approaches peculiar to variability have emerged, such as ConIPF [5], Kumbang [1] and decision-oriented modeling [4]. These approaches introduce a modeling method with constructs for modeling software assets and variability within the assets. In addition, different approaches to modeling variability in existing models of software assets have emerged. For example, orthogonal variability modeling [8] augments existing models with variability specific information. The modeling concepts typically focus on functionality or structure of software.

2.2 Development Management Using Backlogs

The most used agile software development models, Scrum and XP, brought with them a simplified way of recording requirements for products as backlog items in product or sprint backlogs [9] or user stories on story cards [2]. In this paper, we concentrate on the idea and use of backlogs for managing development work and thus in the following paragraphs describe the way Scrum suggests that backlogs are used and managed based on [9] and [10].

A product backlog is a prioritized list of backlog items that must be done to fulfill the product goals. The backlog items in the product backlog can be any kind of items, ranging from bug fixes to transitioning the product to a new technology platform. A product owner controls the product backlog and is responsible for keeping the items and their priorities up-to-date.

When planning each sprint, which is a time-boxed iteration usually lasting 1-4 weeks where an increment of functionality is developed and integrated to the product, the product owner selects backlog items from the product backlog for development during the sprint. Based on these product backlog items, the product owner decides the sprint goals and discusses them and the product backlog items together with the development team. The development team then plans what tasks are needed to get the product backlog items done and to reach the sprint goals. The team estimates the effort needed to complete the tasks and the total estimated effort of all the tasks is compared to the work time the team has available for the sprint. If the total estimated effort needed is bigger than the available time, the development team and the product owner must agree which backlog items and goals are left out of the sprint. When the sprint planning is considered ready, the tasks chosen for the sprint that the team has committed to completing are stored in the sprint backlog.

3. INTEGRATED CONCEPTS

In this section we describe how software product family engineering and backlog management can be integrated in the light of two approaches called Agilefant and Kumbang. Kumbang [1] is a domain ontology (meta-model) and a language to model software product families that is supported with a tool set called KumbangTools[1] [7]. Agilefant [13] is an open source prototype support tool for backlog management that includes managing development as an explicit portfolio of activities involving the developers[2]. In the following we limit the discussion to the concepts that are relevant in this integration. The concepts are summarized in Figure 1.

Kumbang enables modeling the provided functionality and the structure of a software product family including variability. For the purpose of this paper, the main element of Kumbang is that it enables describing product family from feature point of view as a *feature model* (see Figure 1). A *feature* is loosely defined as an end-user visible characteristic of a system. As a means of expressing variability and creating dependencies among features, Kumbang features can be composed of other features. A feature can define any number of *subfeature definitions*, which state what kinds of features can exist under that feature. For example, feature `Dictionary` can specify a subfeature definition `language` with possible features `Finnish, Swedish, English`. If a feature does not define any subfeature definitions, it is termed as

[1]See www.soberit.hut.fi/KumbangTools

[2]See www.agilefant.org for more information

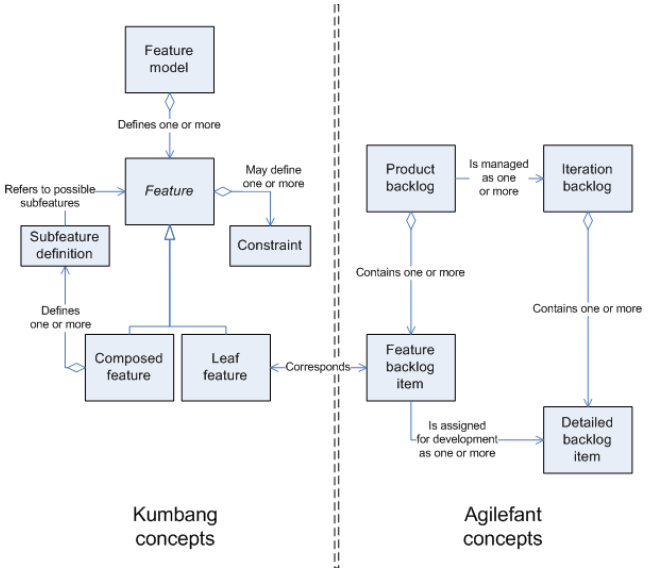

Figure 1: Integrated conceptualisation of Kumbang and Agilefant.

a *leaf feature*, otherwise as a *composed feature*. Further, a feature can define any number of *constraints* that create dependencies to other features. For example, the presence of feature `SpellChecking` may require the presence of feature `Dictionary` in a valid product.

Agilefant, in turn, contains different concepts for backlog management. A *product backlog* is a placeholder for any items referring to a software product family. A special class of these items forms *feature backlog items*. A feature backlog item refers to a planned feature to be developed to the product family. A product backlog consists of one or more feature backlog items. Items of a product backlog are managed as one or more *iteration backlogs*. An iteration backlog groups items under development in each iteration. The feature backlog items are refined and assigned for development as detailed *backlog items*, which can be further refined with development *tasks*. Backlog items have *status* that state, e.g., whether the backlog item is under development or completed. A feature backlog item is considered completed when all its backlog items are completed.

Consequently, the concept of feature backlog item in Agilefant corresponds with the concept of leaf feature in Kumbang feature model. Further, all leaf features of Kumbang model can have a corresponding feature backlog item in Agilefant, and vica versa. This mapping, hence, provides integration between a software product family model and items in a backlog.

4. INTEGRATED TOOL SUPPORT

The concepts of Kumbang and Agilefant as described in Section 3 are supported with tools. In the following we outline capabilities of these tools and describe how to integrate them.

Kumbang is currently supported by two types of tools [7]: *Kumbang Modeler* is a tool for developing configuration models by assisting in defining the types, structures of the types, and constraints; *Kumbang Configurator* is a tool supporting product derivation. A key characteristics for tool support is that the semantics of Kumbang is provided with formal semantics by implementing a translation from Kumbang to a general-purpose knowledge representation language called WCRL (Weight Constraint Rule Language) [11, 1]. This translation enables the use of *smodels* [11] inference engine in supporting tools. Currently, Kumbang Modeler includes functionality to check model validity, i.e. whether there exists a valid product configuration. Kumbang Configurator checks consistency, completeness, and consequences. Consistency means that the product configuration adheres to the configuration model; completeness means that all necessary selections are made; and consequences refers to that the inference engine deduces consequences of variability binding decisions. In addition, further analysis can be developed to KumbangTools.

Agilefant is a prototype support tool for backlog and development portfolio management. Agilefant supports managing multiple backlogs of multiple projects. It helps align development work with business goals by providing different views to different stakeholders. For example, a developer's daily work view summarizes and presents the backlog items assigned to him across all of the projects he participates in. The backlog items with a status of 'started' are displayed on top of the page in order of 1) project priority, and 2) item priority within the project. When development work of a backlog item is finished, the status of the item is changed to 'implemented' and when all the agreed testing is done, the status of the item is changed to 'done'.

We are currently planning implementation of a tool integration such that Kumbang Modeler can be used to define software product family structure, Kumbang Configuration is used to derive valid product configurations, and Agilefant is used for backlog management. The integration between tools is to be based on the correspondence of the concepts: Kumbang Modeler should ensure that all feature backlog items are modeled in the feature model as leaf features; Agilefant should ensure that all leaf features of a Kumbang Model are included as feature backlog items.

5. DISCUSSION

We have outlined the feasibility of integrating product family modeling with development management by combining backlog management of agile development and structural modeling of software product families. More specifically we showed feasibility of integration in terms of Kumbang and Agilefant concepts and tools as examples of software product family modeling and backlog management, respectively. The integration aims towards improving software product family development governance by improved planning, monitoring and controlling of the development work. In this section, we discuss the benefits and applicability of the integration.

The integration *per se* does not introduce additional effort to software development. That is, we showed how both approaches deal with similar issues but from different viewpoints. The integration relies on the existence and correspondence of the similar concept of a feature in both Agilefant and Kumbang. More specifically, such features are the leaf features of a feature model in Kumbang and feature backlog items of Agilefant. Consequently, the integration does not introduce new concepts but rather maps concepts to each other and clarifies how to use the corresponding concept of feature.

On the one hand, the integration provides a means of de-

velopment management in software product families. For example, the status of the features in the feature model can be monitored. This includes checking whether the feature has been completed or is under development. Further, backlogs provide estimates for management such as effort and schedule for features such that completion of a feature can be predicted.

On the other hand, the integration provides a view to the structure of a software product family from a backlog. The view of structure including dependencies seems to benefit, e.g., planning and prioritizing of development. In the aforementioned example, feature `SpellChecking` requires feature `Dictionary` in a valid product. In order to prevent misunderstanding that the functionality provided by `SpellChecking` is achieved only when `SpellChecking` itself is implemented, the dependency between `SpellChecking` and `Dictionary` should be explicitly visible in planning as well. That is, planning can be error prone with backlogs if it relies only on experience or often tacit expert knowledge. With a feature model, the structure becomes more explicit and, hence, increases manageability of planning and accompanying backlog management.

Consequently, the integration seems to provide possibilities for different kinds of analysis. Further, these analyses can even be automated with tools. For example, Kumbang Configurator can be used for checking aforementioned dependencies. However, the need and possibilities for different kinds of analysis needs to be further studied.

Applicability of integration seems not to be limited to the level of leaf features. That is, composed features of Kumbang define possible subfeatures. Hence a composed feature can be analyzed similarly as a leaf feature, for example with respect to status. Analysis of composition is relevant for example when a release consists several technically independent features which only as a whole provide certain practical value to the user. Finally, these extensions to compositions seem to apply even to an entire software product family such that a specific configuration of a product variant or even an entire software product family in terms of all possible configurations can be analyzed.

The integration presented in this paper focused on features. However, the integration can be done to concern other artifacts as well. Kumbang contains concepts for describing a product family from the architectural point of view. In addition, Kumbang Configurator has been extended to cover glue code generation. A feature view and architectural view can be linked by using special implementation constraints. For example, a feature can be implemented by a set of components or more elaborate rules can be specified. Then executable software can be generated from selecting features to the product such that corresponding components are retrieved and glue code to combine them is generated. Similarly, Agilefant contains concepts for, e.g., roadmapping to specify high-level goals of a company. There is also mapping between a roadmap and a product backlog. Therefore, it seems that by taking into account these concepts we can achieve model conformance between various artifacts in software development. However, this is still not fully supported and, hence, extended integration beyond features remains a future work item.

To sum up, integrating structural and development management points of views seem to bring benefits. However, we need to validate the benefits empirically. Nevertheless, based on our experience of case companies, it seems that benefits of combining backlog management and software product family structure could provide solutions to existing problems in the industry.

6. ACKNOWLEDGMENTS

We acknowledge Finnish Funding Agency for Technology and Innovation (Tekes) and the 100-year Foundation of Technology Industries of Finland.

7. REFERENCES

[1] T. Asikainen, T. Männistö, and T. Soininen. Kumbang: A domain ontology for modelling variability in software product families. *Advanced Engineering Informatics*, 21(1):23–40, January 2007.

[2] K. Beck. Embracing change with extreme programming. *Computer*, 32(10):70–77, 1999.

[3] P. Clements and L. Northrop. *Software product lines: practices and patterns*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.

[4] D. Dhungana, R. Rabiser, and P. Grunbacher. Decision-oriented modeling of product line architectures. In *WICSA '07: Proceedings of the Sixth Working IEEE/IFIP Conference on Software Architecture*, page 22, Washington, DC, USA, 2007. IEEE Computer Society.

[5] L. Hotz, K. Wolter, T. Krebs, S. Deelstra, M. Sinnema, J. Nijhuis, and J. MacGregor. *Configuration in Industrial Product Families*. IOS Press, 2006.

[6] K. Kang, S. Cohen, J. Hess, W. Novak, and S. Peterson. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90-TR-21, SEI, 1990.

[7] V. Myllärniemi, M. Raatikainen, and T. Männistö. Kumbang tools. In *Software Product Line Conference*, 2007.

[8] K. Pohl, G. Böckle, and F. van der Linden. *Software Product Line Engineering: Foundations, Principles, and Techniques*. Springer, 2005.

[9] L. Rising and N. S. Janoff. The scrum software development process for small teams. *IEEE Softw.*, 17(4):26–32, 2000.

[10] K. Schwaber and M. Beedle. *Agile software development with Scrum*. Prentice Hall, Upper Saddle River, 2002.

[11] P. Simons, I. Niemelä, and T. Soininen. Extending and implementing the stable model semantics. *Artificial Intelligence*, 138(1-2):181–234, 2002.

[12] M. Svahnberg, J. van Gurp, and J. Bosch. A taxonomy of variability realization techniques. *Software — Practice and Experience*, 35(8):705–754, 2005.

[13] J. Vähäniitty. Do small software companies need portfolio management? In *Proceedings of the 13th International Product Development Management Conference*, 2006.