Facing Future Software Engineering Challenges by Means of Software Product Lines

David Gollasch

TU Dresden, Faculty of Computer Science, Institute of Software- and Multimedia-Technology, Chair for Software Technology
david.gollasch@tu-dresden.de

Abstract. As software requirements change very rapidly, software engineering principles have to keep up with these changes. This paper names significant trends in the field of software engineering and discusses ways to cope with these trends in the future. The software product line approach seems to be promising here to have a strong influence in the future on larger software products where it's affordable to apply the product line approach. While it currently allows abstraction of variability in space, it will support variability in time as well in the near future. This paper discusses in how far this approach can cope with the former identified upcoming trends.

Keywords: software engineering, future trends, software product lines, software evolution, hyper feature models, variability

1 Introduction

Software engineering is the field of creating and maintaining software products. As the requirements and expectations on software are subject to change very rapidly over time, this has a strong influence on methods, tools and processes in the software engineering field. Hence, it is meaningful to discuss upcoming trends in this area in order to find feasible development approaches.

One promising approach to cope with the rising challenges is software product line engineering. Software product lines allow efficiently developing complex software products by applying reuse-principles and introduce a high level of abstraction at design time. It is the goal of this paper to discuss the product line approach in context of its future-awareness. Therefore, the concept of hyper feature models will be discussed.

To obtain this goal, section 2 of this paper discusses currently observable trends in the area of software engineering according to their influence in the future. This leads to new challenges in the future that new software engineering approaches have to face. As software product lines appear to be promising, section 3 presents the underlying principles and aspects to address the challenges. The concept of hyper feature models as current state-of-the-art findings is presented as well, as this is highly interesting for the future-readiness of the

presented development approach. Section 4 sets the product line approach of section 3 including the hyper feature model approach against the discussed trends of section 2. Finally, section 5 summarizes the findings and gives a short outlook for further investigation.

2 Upcoming Trends in Software Engineering

There are some observable trends in today's field of software engineering. Boehm identified ten of them in his findings [1]. This paper focusses on a selection of five out of them that I assume to be relevant for the further discussion.

More rapid development and adaptability. As software requirements change increasingly fast, the development has to keep up. Therefore, future development strategies have to enable a more efficient way of composing software and adapt to changing requirements.

More software criticality and need for assurance. Software is becoming more and more ubiquitous as it is easier to adapt software to new requirements than hardware. This leads to the need of a higher software quality. Reliable software is indispensable in some fields, e.g. logistics and medicine.

Increasing complexity and need for scalability and interoperability. Due to the rising importance of the internet and networks in general, software that interacts with other systems is increasingly required. That leads to a growth in complexity and to software that can be scaled.

The need to accommodate commercial off-the-shelf and legacy software. The interoperability between a product and existing COTS (commercial off-the-shelf) or legacy software that is already in use on the customer's side gains in importance as well.

Stronger accentuation on users and end value. Technical borders that developers have to cope with are becoming less relevant in software engineering. That more and more moves the user and cost-benefits into the focus. Future software development strategies should support this trend by means of enabling the composition of software that fits the user's requirements as exactly as possible.

2.1 Coping With These Trends

The above-mentioned trends are on a less technical level, but rather on a relatively abstract one.

Generally, it is the developer's goal to create software with the right set of features, that can be developed efficiently and that can evolve easily. This implies

that coping with these trends is only possible if software engineering can be done on a more abstract level [6]. Higher abstraction allows developers to focus rather on the conceptual aspects of software than on the technical ones. This is possible as abstraction means the reduction of the concrete software to a more general conception.

A common way to introduce additional levels of abstraction is the use of models. In consequence, a model based engineering approach can help to cope with the mentioned trends. A low level of abstraction means less generalization. A typical low abstraction level of software is a complete structural model of the code in form of a UML class diagram. A higher level of abstraction can be a component model or even the representation of features in feature models.

3 Software Product Lines and Feature Modeling

Software product line engineering allows the efficient development of relatively similar software products due to capturing commonalities and variable functionality. This allows an intense level of reusing software components in multiple products that can be derived out of one product line.

To illustrate the principle of software product lines it is feasible to find a relation to building blocks. There are two development life cycles: the *Domain Engineering* life cycle and the *Application Engineering* life cycle [7]. The main purpose of the first life cycle is to build up the actual product line which includes the development of all relevant parts or aspects of the later to be derived products. These parts and aspects can be interpreted as a set of different building blocks. During the second life cycle the actual product will be derived which can be seen as the process of building a specific object out of the former prepared set of building blocks. Blocks that are part of every derived object define *commonalities*; blocks that are not mandatory for each object define *variability*.

The model-based product line approach allows the mentioned level of abstraction to cope with the challenges stated in section 2. That makes it appear to be a very promising approach [2].

Feature models are used to represent product lines through a set of features and dependencies between those features. Features can map to structural and behavioral (functional) and/or qualitative and characteristic (non-functional) aspects of a software system [5].

A concrete product can be derived out of a product line through a configuration process. This process encompasses the selection of features included in the product line while respecting existing constraints.

3.1 Variability in Space

Software product lines allow multiple levels of abstraction as features in feature models can represent functional and non-functional aspects of a software. So the levels of abstraction can be determined as follows (according to [5]):

1. structural level (representation of functional aspects of a software)

2. (user-visible) behavioral level
3. qualitative/non-functional level
4. software-system's characteristic level

These levels of abstraction have the structure of resulting configurations in mind. These variable aspects are also called "variability in space". This can be assumed as state-of-the-art.

3.2 Variability in Time

Current product lines only support variability in space. There is a lack of supporting evolutionary processes. This is important in terms of the support for future-ready software development. Respecting software evolution in product lines is called "variability in time" which introduces the time abstraction level.

Variability in time enhances reuse in product lines which makes it a necessity when it comes to future-ready software development.

Elsner et al. [3] identify three types of variability in time:

1. the linear change over time (evolution and maintenance)
2. supporting multiple versions at one point in time (as part of a configuration)
3. binding over time where different types of variability are present at different stages of development

Referring to the second type of variability in time, the support of multiple versions in configurations, there already is one approach to face this. Seidl at al. [8] propose the use of hyper feature models. They describe an extension of feature models in a way that supports maintaining multiple versions of a feature in a product line.

The field of supporting evolutionary aspects of product lines is the objective of current and ongoing research. In terms of coping with upcoming trends in software engineering, this could be a key aspect in making the software product line approach a feasible solution and having a massive impact in future software development processes as stated by Boehm [1].

3.3 Hyper Feature Models

One approach to include version-awareness into the software product line approach is the *hyper feature model* approach introduced by Seidl et al. [8] as an extension to "normal" feature models introduced by Kang et al. [5].

A feature model represents the *commonalities* and *variabilities* of all software products of a product line. Therefore, it includes a set of features and the dependencies between them.

A feature model is structured as an acyclic graph respectively a hierarchical tree structure. So each feature (except the root feature) has one parent feature and can have multiple sub-features. A concrete product of the product line is called *configuration* and describes a valid subset of the product line's set of

features. One rule that ensures the validity of a configuration is that the selection of a feature automatically leads to the selection of its parent feature.

Next to the hierarchical structure, there are cross-tree-constraints. These constraints define additional rules for the validity of configurations. They represent rules like "If feature A is part of the configuration, feature B has to be selected as well" or "Feature A and B cannot be selected at the same time in one configuration".

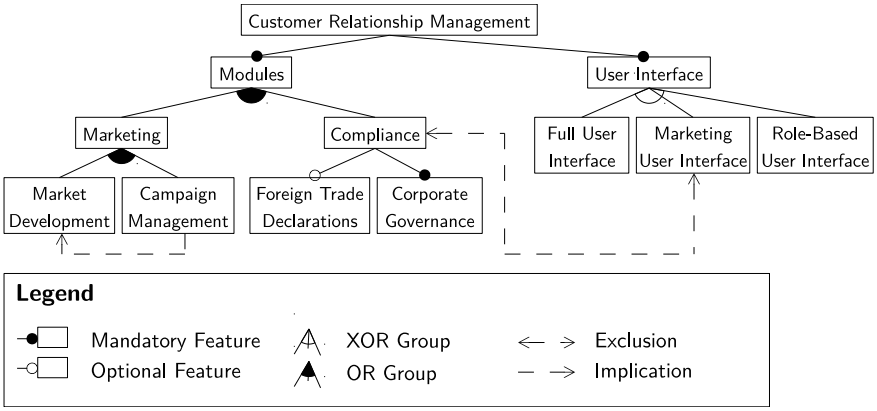

Fig. 1. Example of a feature model based on [4, p. 10]

Figure 1 shows a small example of a simple feature model with all essential elements of such models. The illustration shows the hierarchical structure of features, beginning with the root called *Customer Relationship Management*. During the configuration process, features have to be selected from top to bottom, starting with the root feature and respecting the given dependencies.

Hyper feature models additionally allow maintaining multiple versions of each feature into the model, including the representation of development branches (so each version can have a predecessor and one or multiple successors). The available cross-tree-constraints are extended by version requirements, e.g. "If version 2.0 and above of feature A is part of the configuration, feature B has to be selected in version 1.1 and above as well." Figure 2 extends the example of Figure 1 to an hyper feature model. Each feature has been extended by multiple versions and development branches. Version-aware cross-tree-constraints are not represented in this example.

3.4 Risks, Challenges and Limits of Software Product Lines

The software product line approach is rather complex and needs strong governance. Model inconsistencies directly lead to errors while generating valid configurations and concrete software products. Each change in code and model has to be reflected on each other.

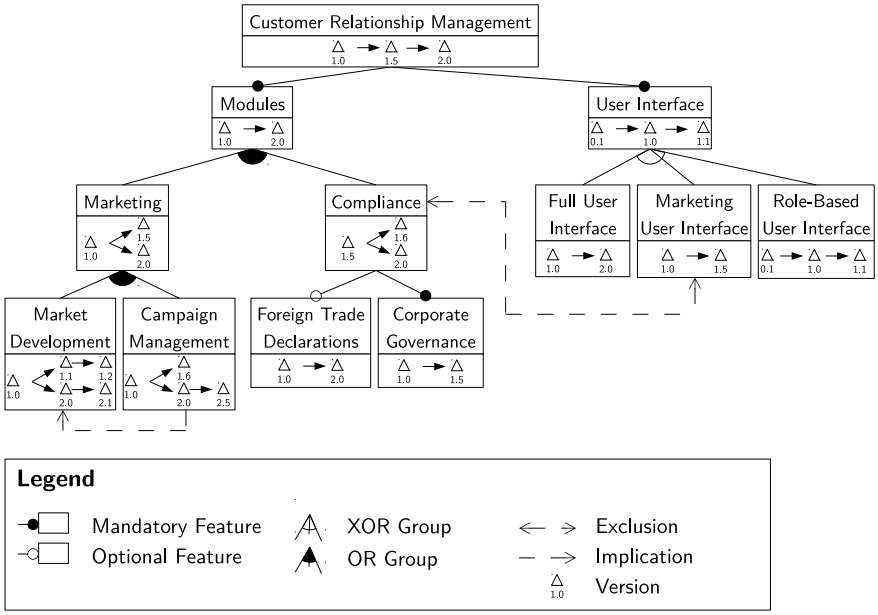

Fig. 2. Example of an hyper feature model based on Figure 1 and the notation proposal of Seidl et al. [8]

That makes is costly and inefficient if only one concrete product should be developed. The software product line approach is only affordable and saves time if multiple products should be generated out of one product line.

Nevertheless, the link between source code and the features in the model is fragile and needs a careful maintenance to ensure the seamless generation of concrete, running software products. The risk of generating redundancy and architecture erosion should be kept in mind.

4 Facing Upcoming Trends Through Software Product Lines

Hyper feature models are only a first attempt to introduce a time abstraction level in software product line engineering. But if we assume that software product lines support variability in space and time, it may cope with the upcoming trends as follows.

According to the trend of a *rapid development*, the strong use of reusing principles helps facing this trend. As a software product line and ecosystem can become very complex due to plenty of dependencies, version-awareness simplifies reusing heavily. However, the product line approach is only appropriate when it is the attempt to generate multiple products out of a product line. That makes it inappropriate for a lot of other software development projects where only one single piece of software is the desired result.

To address the *need for assurance*, composing software out of reliable and proven components can face this trend. As those components are never newly implemented, a version-aware product line will be important as well.

Software product lines cope with *complexity* due to the reuse principle. I do not see a direct advantage in the additional variability in time.

The *need to accommodate COTS* makes it important to keep the own software compatible. As there is no way to influence the life cycle of the third-party software component, version-awareness is fundamental.

The product line approach *copes with user-specific configurations* due to the general variability principle. I, again, do not see a direct advantage in the additional time abstraction.

5 Conclusion

Software in the future tends to get more and more complex and user-oriented while there is a need to get them developed faster without a loss in quality. These trends are a major challenge for software engineers in the future.

Abstraction is one possible key to success. The software product line approach as a model-driven development strategy already allows abstraction on the structural, behavioral, qualitative and characteristic level in form of variability in space.

Variability in time as a time abstraction level will be necessary to cope with the mentioned trends in the future considering evolutionary processes. One approach is the use of hyper feature models as they allow maintaining multiple versions of each feature in one product line.

This makes the software product line approach very promising to be the appropriate development methodology for many software projects in the future. As this approach is very pricey to apply, it is only affordable if multiple software products should be generated out of one single product line. Otherwise, the overall development process needs too much time and is rather inefficient.

To get over this drawback, it would be interesting in how far it is possible to combine agile development strategies with the presented software product line approach. If there is a feasible solution, it would be potentially possible to combine the advantages of both worlds and get over the disadvantages mentioned.

References

[1] Barry Boehm. “Some Future Software Engineering Opportunities and Challenges”. In: *The Future of Software Engineering*. Ed. by Sebastian Nanz. Springer Berlin Heidelberg, 2011, pp. 1–32.

[2] Manfred Broy. “Seamless Method- and Model-based Software and Systems Engineering”. In: *The Future of Software Engineering*. Ed. by Sebastian Nanz. Springer Berlin Heidelberg, 2011, pp. 33–47.

[3] Christoph Elsner, Goetz Botterweck, Daniel Lohmann, and Wolfgang Schroeder-Preikschat. “Variability in Time - Product Line Variability and Evolution Revised”. In: *Fourth International Workshop on Variability Modelling of Software-intensive Systems*. VaMoS. Vol. 10. Linz, 2011, pp. 131–137.

[4] David Gollasch. “Qualitaetssicherung mittels Feature-Modellen”. Bachelor Thesis. Dresden: TU Dresden, 2013.

[5] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson. *Feature-Oriented Domain Analysis (FODA) Feasibility Study*. Carnegie-Mellon University Software Engineering Institute, Nov. 1990.

[6] K.RustanM. Leino. “Tools and Behavioral Abstraction: A Direction for Software Engineering”. In: *The Future of Software Engineering*. Ed. by Sebastian Nanz. Springer Berlin Heidelberg, 2011, pp. 115–124.

[7] Frank van der Linden. *Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering*. In collab. with Klaus Schmid and Eelco Rommes. Berlin ; New York: Springer, 2007. 333 pp.

[8] Christoph Seidl, Ina Schaefer, and Uwe Assmann. “Capturing variability in space and time with hyper feature models”. In: ACM Press, 2014, pp. 1–8.