Using a Quantitative Approach for Defining Generic Components: A Case Study

Peter Knauber and Klaus Schmid

Fraunhofer Institute for
Experimental Software Engineering (IESE)
Sauerwiesen 6
D-67661 Kaiserslautern, Germany
++49 (0) 6301 707 242,158
{Peter.Knauber, Klaus.Schmid}@iese.fhg.de

ABSTRACT
In this paper we give a sketch of a new quantitative approach which can be used to identify and cluster system functionality that should be developed in a reusable fashion, for example, when starting product line development. We demonstrate the applicability of the approach and its validity in a case study.

Topic Areas:
Case Studies and Experiences, Product Line Architectures, Architecture assessment and reviews

1 INTRODUCTION

Product line development is a new approach to software mass production. It focuses on the notion of a product line, i.e., a set of non-identical, but strongly related (in terms of functionality) systems [1]. The key enabler for this approach is to derive a software architecture which appropriately supports the products which ought to be developed. A core goal of this software architecture is to encapsulate exactly the functionality that is needed reusable in generic components. In this paper our focus is on an approach for determining which components to package in a reusable manner. The approach we used originates from product line scoping and aims at the identification of system functionality that should be developed for reuse based on a detailed assessment of the costs and benefits linked to developing this piece of functionality for reuse.

In our approach there is no constraint on the level of detail at which the system functionality is defined. It can range from high-level requirements down-to actual component specifications. In the case study we report upon here we used high-level system requirements as a starting point and wanted to know which are the specific functionalities that should be generically packaged. The results of this analysis can be regarded as “reusability requirements”.

The approach we used [2] is a quantitative approach that is tailorable to the specific business needs of a company. Thus the approach provides numerical evaluations of the benefit (relative to the business objectives) of having certain functionality in an easily reusable manner. This allows to use this information for making trade-off decisions during architecting [4], e.g., if two features cannot be simultaneously well-packaged for reuse.

The position we put forward in this paper can be briefly summarized as:

> Reusability in the software development process is mainly a function of the architecture. However, the priorities assigned to making certain functionalities reusable should be based on a detailed quantitative evaluation of these functionalities.

We show how this can be done using a case study we performed with an industrial partner.

2 CONTEXT

The case study we report on in this paper was performed with MARKET MAKER a leading player in the market of software for stock market investment and information services for non-professionals in Germany[1]. At the time we started the cooperation in autumn 1998, there were five fully employed software developers supported by several part-time employees working on four products:

- *Product A* is the high-end product which is used by professional brokers working in banks and semi-professionals. This product has the most advanced and most complete functionality of the MARKET MAKER products. It is used to download stock market data (off-line) provided from MARKET MAKER, to manage it, and to evaluate it according to various pre-defined and user-definable functions.
- *Product B* and *Product C* are smaller versions of *Product A* for non-professional users with less functionality. They are distributed by other companies which also provide hotline services for them under a different label.
- *Product D* is used by people working on-line at the stock market: Via satellite changes in ratings of shareholder values, opening and closing rates, and important messages were sent, were decoded by the product and could be displayed using customizable views.

The people developing and maintaining the products were all very experienced, most of them working for several years (at least part-time) in the domain. Thus, special documents describing the domain, the architecture, or the design of the products were at the point in time of our study not needed and hardly available. But contrary to our experience in many other projects, we found that there was a common understanding of the domain and the product. All the developers could be called domain experts, as they had a lot of experience in developing the system. However, this fact was expected to change during the next years: The company intended to produce many more products and to grow significantly in terms of developers. These would not at all be experienced, neither in the stock market domain nor with the software. Thus, problems were expected to teach these new people about the domain and the products and to introduce them into the existing development team without written documentation.

Facing such problems it was decided to analyze the potential of going towards product line development. As the resources were tight, it was clear that the case study had to be performed with the least possible effort, even though it was obvious that a reunified, common kernel for the products would decrease the effort for maintenance and evolution of these products significantly. But documents describing the architecture of the products, their components and interdependencies, were not available. Consequently, a reverse engineering of the as-is architecture would have introduced too much effort before the actual decision about which components should be reunified or be newly developed in a reusable fashion and which other product parts would be affected by them could really be made.

The characteristics of this situation: extremely scarce resources, already existing systems, highly qualified experts, and true reverse engineering requiring too much effort, dictated the way we had to go: we had to analyze the situation with a low impact approach, that would not require any information on the existing code. We describe this approach in the following section.

For us, this situation had also the benefit that, as already systems were built and we had experts available, we could

1. The project was partially funded by the Ministry of Economic Affairs, Transportation, Agriculture, and Viniculture of the state Rhineland-Palatinate, Germany.

Goals for product line development	Weighting
Reduction of Time-to-market — it should be possible to reduce the development time for new products	0,637
Effort reduction — the overall effort needed for developing new products should be reduced	0,047
Quality improvement — the reuse base should help to improve quality in terms of reliability, performance, etc.	0,093
Functionality extension — it should be possible to equip new products easily with a large range of functionality	0,223

Table 1: Product line development goals and weightings

have our results validated by experts.

3 APPROACH

An earlier version of the approach we used for identifying generic functionality has been described in [2]. The basic structure is as follows: first the relevant goals the company is pursuing with a product line approach are elicited. Table 1 describes the goals and the respective weightings that were assigned to them by the customer in the case study. Simultaneously we elicited the relevant functionalities of the systems. Due to resource restrictions we had to confine our evaluation to a sub-part of the complete functionality. The various goals were refined into metrics that could be used for deriving a benefit (cf. Table 2). This approach to metrics development is the core step where adaptation of the approach to the specific environment and business goals happens (cf. [3]). Note, that the only metric which looked into the existing implementations was “already implemented”. Apart from this, no code-related information was used, as the “interaction” metric refers to the conceptual relationship with other features. Thus, this is also globally assessed, i.e., independent of the specific products.

Using these metrics the various features can be characterized and the benefits derived from them can be computed. Table 3 shows an excerpt from the resulting product map. The evaluation gives explicit values corresponding to the benefit of having the specific functionality. Based on similarity among the functions, the features were already clustered into groups. These clusters do not necessarily correspond to similar evaluations. Sometimes a single feature received a low score, while the rest of the group received a rather high score.

The column “implemented” in the product map (cf. Table 3) marks which function has to be deployed in which product. Additionally, features that have a particularly high/low benefit score are colored. This allows to form clusters that should be packaged together (note that this is not necessarily identical to the groups of features that were identified as being related on the conceptual level by the experts). The clusters were defined to package together features that had related functionality and that had similar benefit scores. However, in some cases features were added to groups despite having different scores, in order to derive larger coherent groups. This was then performed based on the relationship among features in the group and the coherency of the deployment of features across the product range.

Depending on the height of the score and the interrelation of features the following recommendations were given:

- A generic *implementation* of <*function group*> should be made available

Metric	Description
interaction (global)	how strongly does an implementation of the feature interact with other feature implementations?
already implemented (per system)	was the feature already implemented in any product?
priority (per system)	what is the priority of having this feature in this system?
effort (per system)	what is the effort of implementing this feature in the context of this system?

Table 2: Metrics used in the case study

Product Characterization / Features		Interaction	existing products				Future products				score
			Product A			...	Product E			...	
			priority	implemented	effort		priority	implemented	effort		
Time series	line chart	1	4	X	1		4	X	1		0.14285
	dot chart	1	4	X	1		4	X	1		0.14285
	histogram	1	4	X	1		4	X	1		0.14285
	...										
Signals											
signal sources	Indicators	3	4	X	2		3	—	2		0.4
	Trend-Analysis	3	4	X	1		0	—	1		0.10714
	Tools	3	2	X	0		0	—	0		0.13392
	Trading systems	3	0	—	2		0	—	2		0.07142
	...										

= strongly recommended = not recommended at all

Table 3. A Product Map Excerpt

- A generic *implementation* of <*function group*> should be considered
- In case of further products using <*function group*> a generic *implementation* of <*function group*> should be developed
- A generic *interface* to <*function group*> is strongly needed
- A generic *interface* to <*function group*> is needed
- Generic support for <*function group*> is currently not useful

The recommendations were based on the following rules:

- Generic implementations were recommended, when:
  - considerable benefit can be accrued for further to be developed products (level of recommendation depended on the value of the score)
- Generic interfaces were recommended, when both:
  - further implementation of a feature was needed, especially in cases where the current interaction of the feature with its environment was regarded as high and
  - the benefit of having a reusable implementation was not regarded as high

4 EVALUATION

A list of these recommendations was developed and given to the experts. Additionally, we asked for the cases where already corresponding changes had been performed. For answering the questions an agreement scale was used with the values:

- I do absolutely agree
- I do mostly agree
- Neutral
- I do mostly disagree

I do absolutely disagreeThe resulting ratings showed a high agreement with the packagings we proposed (cf. Table 4). Further, asking whether the corresponding changes had already been performed gave a very interesting picture:

In 25 out of the 31 cases the ratings for “the genericity level this should have” and “the genericity that has been already implemented” differed by at most 1. In the remaining cases there was three times a difference of two levels and three times a difference of three levels. Always the expert was positive about the need for additional genericity but less about the existing genericity. In particular the core expert rated three times: “I do mostly agree” to our proposed level of genericity, while rating “I do absolutely disagree” to “this is already generically implemented”.

.Summarizing, our approach provided genericity evaluations which fitted extremely well the level of genericity that the developers thought was needed. When analyzing this data one should keep in mind that the developers did built their evaluation on several years of development effort, while we did built our evaluation merely on metrics, to which – except for “already implemented” – one could assign values without any system knowledge (the metric “interaction” was based on the conceptual interrelation of features and not on implementation-specific aspects). Given this background we thought it to be rather surprising that we could actually spot some areas that are still – after several years of development – a major concern to the developers and thus help them to focus on certain areas for structural improvement of their software.

Agreement Value	Count	%
I do absolutely agree	8	28
I do mostly agree	17	59
Neutral	3	10
I do mostly disagree	1	3
I do absolutely disagree	—	0

Table 4. Distribution of Agreement Values

5 CONCLUSION

In this paper we gave a sketch of a new quantitative approach which can be used to identify and cluster system functionality that should be developed in a reusable fashion. Identifying functionality that should be made reusable is a task, which always has to be performed in the context of product line development. We found the approach extremely helpful:

- it can be applied without having existing systems or an explicit system architecture, making it applicable in very early phases of (product line) development,
- it can be applied on varying levels of detail with the expectation to produce even more precise results when feeding it with more precise input, and
- it produces reliable results while requiring little effort.

In the case study described in this paper, we were able to prove the applicability of the approach and to validate its results. With a very high reliability it pointed out the areas which were considered as being important to be developed for reuse by experienced domain and system experts.

References

[1] J. Bayer, O. Flege, P. Knauber, R. Laqua, D. Muthig, K. Schmid, T. Widen, and J.-M. DeBaud. *PuLSE: A methodology to develop software product lines*, in Proceedings of Symposium on Software Reusability’99 (SSR’99), Los Angeles, CA, USA, pp. 122-131, 1999.

[2] J.-M. DeBaud and K. Schmid. *A Systematic Approach to Derive the Scope of Software Product Lines.* International Conference on Software Engineering (ICSE’21), Los Angeles, CA, USA, pp. 34–43, 1999.

[3] K. Schmid. *An Economic Perspective on Product Line Software Development.* First Workshop on Economics-Driven Software Engineering Research, Los Angeles, CA, USA, 1999.

[4] R. Kazman, M. Barbacci, M. Klein, S.J. Carriere, and S.G. Woods. *Experience with Performing Architecture Tradeoff Analysis*, International Conference on Software Engineering (ICSE 99), Los Angeles, CA, USA, pp. 54-63, 1999.