1

Tools for Design Rationale Documentation in the Development of a Product Family

J.Savolainen
Helsinki University of Technology
P.O. Box 9700, 02015 HUT, Finland
Tel+358 9 451 5135, Fax+358 9 451 5351
Juha.Savolainen@hut.fi

Abstract

Building a product family is a complex task that requires careful management. To be successful in this development one has to be able to satisfy requirements for all the different members of the product family. In order to support this task we present a unified method integrating two different ways to describe software architectures. Our approach supports documenting design rationale in a complete way allowing the design to evolve gracefully. This is achieved by presenting all the design steps in a design decision tree and simultaneously mapping variation of system characteristics in a requirement space.

Keywords

Software Architecture, Product Family, Requirement Space, Design Decision

---

This work is carried out in the ABACI project. ABACI project is a co-operation project between Helsinki University of Technology and NMP, NRC, NTC and Vaisala. ABACI is supported by Tekes (Technology Development Centre Finland).

1 INTRODUCTION

Global marketplace has placed companies in a situation where they have to create products extremely fast. This development should be cost-effective and produce high quality systems, which poses functionality needed by the customer. However, the complexity of systems is constantly increasing and the number of product variations needed to cover the marketplace is high.

Current research on software engineering has explored possibilities of using product lines as a tool to achieve organised re-use. Product lines and families have demonstrated improvements in quality, development cost and time-to-market attributes. However, the true potential of product lines has not been realised.

Re-use has been considered as a solution that can help companies to reduce development time and achieve higher quality. Product families are one of the most promising re-use techniques. A company that is producing many products which all share similar properties can structure these systems as members of a product family. Software doesn't just happen to be reusable or acceptable for the product family. Creating a product family requires careful planning, a complete domain analysis and structured design. None of these tasks is trivial, so there is clear need for tools supporting this kind of development.

Design decision trees and requirement space can be used to create product families. Our approach leads to a structured and organised design, which is easy to maintain and modify. This method maximises benefits available from the product family approach by re-using the design of individual members of the product family.

2 PRESENTING DESING RATIONALE

Requirement analysis and documentation is an important part of designing a software system. Without a complete analysis process it is just pure luck whether the system meets customer's requirements. Demand for the quality of the requirement analysis process becomes even more important when we are designing a product family. One must now consider all the requirements of all the members of the product family.

Designers need tools to help them during the analysis and design process, and a method to communicate this knowledge to persons responsible for the evolution of the system. We propose an approach where two different tools are combined to provide support for the whole software lifecycle. In the following chapters we present these tools and discuss how they can be used to help designers and maintenance personnel.

2.1 Design Decision Trees

We use Design Decision Trees (DDT) to incrementally document, refine, organise and reuse knowledge for software design. Design Decision Tree is a partial

ordering of design decisions put in the context of incrementally specified problem requirements and the constraints imposed by earlier decision [Ran & Kuusela, 1996]. Each node of DDT documents a design decision and rationale for it. Arcs between the nodes reflect the dependency between the decisions. This dependency provides us a way to relate developer's decisions to original requirements.

2.2 Requirement Space

In real software projects there is a lot of pressure put on a developer. The developer should be able to satisfy strict time deadlines in addition of many functional and non-functional requirements. However, there are always much more requirements than any single software project can manage. There is a clear need for prioritisation among the requirements. In our product line development project we have used a cost-value approach for prioritising requirements [Karlsson, 1997].

We propose a simplified notation to capture all the requirements for a product family. The approach is based on an idea of pair-wise requirements comparison. A requirement is represented as a line of variation, where the total variation is a sum of the requirement variation of all current and planned members of a product family. The line shows total variation of a value respect to the requirement. Pair-wise variation comparison is a pleasant way to represent requirements for the product line to be developed.

Variance is represented as an area, not as a point. The area represents a place where the properties of final designs would lay if every branch of the DDT after the current decision would be implemented. The area of the decision is the range of functionality where different designs cluster on the requirement space. This unpredictability comes from using patterns to describe DDT nodes; there are still many ways to implement a system. In our approach we make more general decisions first, causing the area in the requirement space to decrease after every decision.

The visualisation of variance is important for the development of the product family. Variance in a requirement represents a dimension, which the product family must support. In the product family there has to be enough flexibility in respect to this dimension. Developing a framework for the product family requires knowledge of what is going to be changed during the system's lifecycle.

Variability can be supported be enabling flexibility for the dimension. Our process can handle only planned variance – the variance that can be predicted during the design of the product family. However, it should be noted that in our approach, ordering design decision in a way that more general decisions are taken first maintains maximum flexibility in respect to important requirements. But in any case, if a completely new requirement emerges, there is no way to support such variance.

3 SUPPORTING PRODUCT FAMILY DEVELOPMENT

A lot of work is done on a software product after it is deployed. It is a major requirement to have a design that supports effective maintenance. In our approach we are addressing two major issues in the software lifecycle – evolution and maintenance. Successful product line approach requires that design rationale be presented in an easily accessible way. This rational is essential to provide context for a maintenance engineer who is responsible for a system's evolution. He needs to understand the design decisions and all the constraints associated with the product line architecture.

The product line architecture is derived from requirements in its domain. To allow adequate support for evolution, we must describe the architecture and also the reasons for making the architectural design decisions. A solution must also model the requirements in a way that each architectural decision can be derived from the requirements. The effect of the decision can be seen in some way reflected in the functionality of the system and also as a partial fulfilment of the original requirement.

Our work provides just that. Documenting a design in a design decision tree allows a structured presentation of design information. On the other hand, the requirement space captures all the requirements for a system and allows a visualisation of these requirements. This method also shows the effect of a single decision on the functionality of the system.

In our model we assume that all the decisions are taken one after another in a way that more general design decisions are made first. Now we have incrementally specified decisions, which relate directly to incrementally specified functionality. This functionality becomes more defined in every design cycle and approaches the original requirements of the system. This continuous, incremental specialisation is documented in a way that can be easily accessed by maintenance engineers.

Visualisation of how the functionality of the system approaches the original requirement in respect of the design decisions made during the development of the system has a great value. Producing a new member of the product family becomes a task of finding a branch in the DDT that satisfies the requirements associated with the system. An automated visual builder that allows generating products from its specification can support this process if systems are in the range of common properties of the product family. This approach has been used in a case of framework design [Roberts & Johnson, 1996].

The common requirements of the product family give limits to the modifiability of the system. If requirements that do not fit into these limits emerge, then we may have to redesign the common framework.

4 WEATHER STATION CASE STUDY

The common requirements of a product family define the core functionality for our framework. These requirements are common assumptions shared by members of the product family, so we can start our development process of the framework by incrementally satisfying these common requirements.

Consider a measurement collecting and processing equipment such as a weather station. Weather stations run on various hardware and operating system platforms. They perform numerous calculations and broadcast different kinds of reports to various terminals by a serial cable or via a satellite link. Due to the nature of weather observation domain there are multiple products needed to cover the market place. There are different user demands and regulations, which must be included in the product functionality. In this kind of environment high adaptability to various hardware platforms is a key requirement. Also high performance must be achieved using only limited hardware resources. Simple requirement space is shown in figure 1.

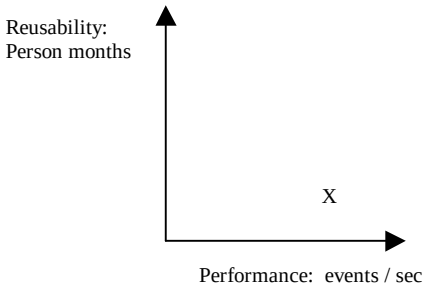

Figure 1 A requirement space for the weather station

There are two common methods to create the framework for a product family. The first method produces a framework that is a sum of all variance associated with planned product family. However, this creates a kind of framework that often cannot evolve because both the common properties and the variance are included into the framework. The second method includes only the common partition of the product family in the framework and uses separate components to support the variance among the members of the product family. This allows new products to be included in the product family if they share the same common assumptions. However, this will use only a portion of the total re-use potential. Often members of the product family can be divided into groups, which have similar properties. When we encapsulate variance into separate components it's complicated to re-use design because we cannot use the component as it is. By using design decision trees and requirement spaces we can support also this kind of re-use by documenting the decisions made during the component design phase.

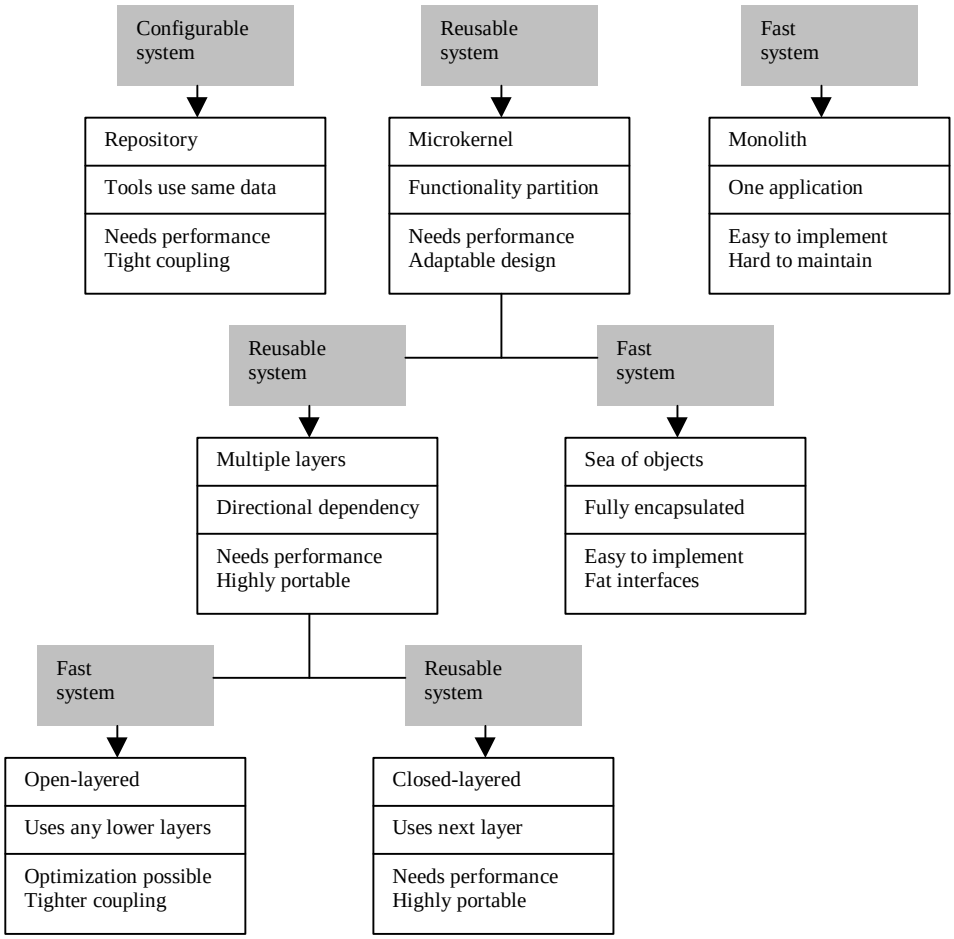

Figure 2 Design decision tree for the common framework of the weather station

Figure 2 shows the design decision tree of the weather station product family. It displays only the common portion of the framework. However, similar process can be used to create a new product family member. If we have documented every product family member in our DDT, we can use similar process to create a new product. Using the requirement space we can directly see, which existing family member has most similar requirements.

Now a developer can start a design process by tracing design decisions backward from the leaf node of the similar product towards to the root of the DDT. He continues upward until enough flexibility is gained to satisfy the requirements of the new system. In this way we can achieve re-use also outside the boundaries of the common framework, by re-using also design of one product family member. Figure 3 displays the requirement space describing the DDT shown previously.

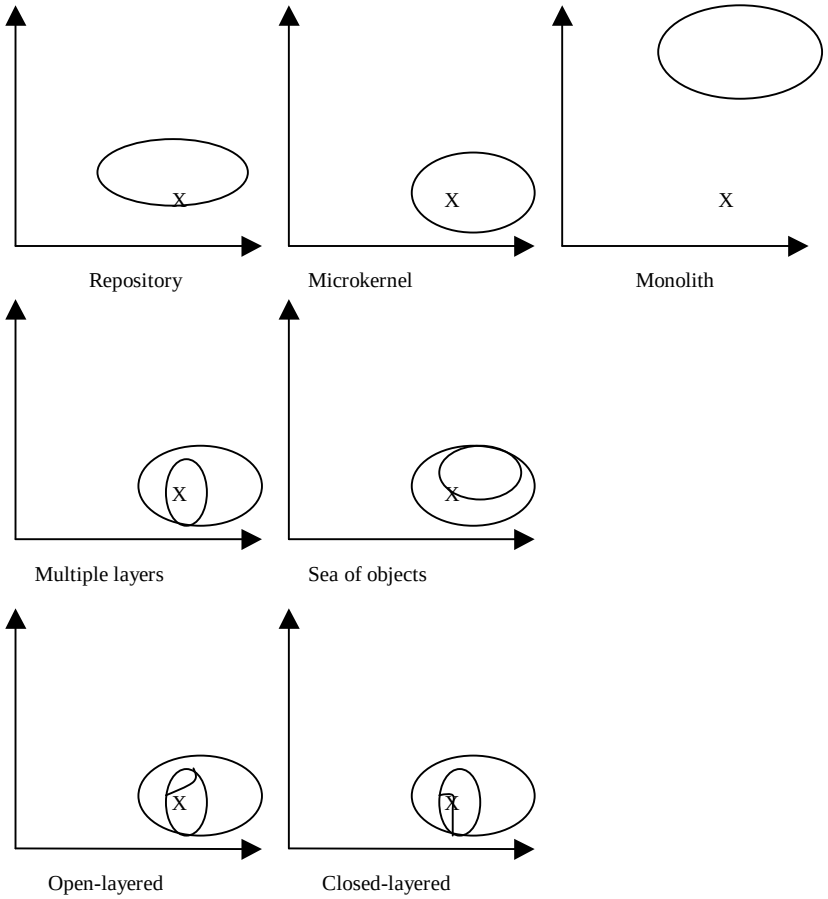

Figure 3 The requirement space for the weather station

5 RELATED WORK

When designing a software system, satisfying requirements is crucial for the success of the project. Representing design requirements in some way is done in almost every project. Many authors have used the idea of requirement comparison and some have also visualised results by plotting them on a plane.

Lane has used a design space to organise and capture design knowledge by identifying functional and structural choices [1990a]. Capturing functional requirements and creating design rules based on them can be used to help designers to develop software systems that will meet user's functional requirements. The process of finding correlation between different design space dimensions relays on experience gained during a complete analysis of the application domain. Lane has used this approach for user interface software architectures [1990b].

Alonso and colleagues have proposed a framework solution for documenting design decisions in product family development [1997].

Product families have been designed using domain analysis to find out commonality and variability of the product family. A good example of this approach is a commonality analysis method (FAST) by Ardis and Weiss [1997]. The FAST method's commonality analysis gathers all the assumptions true for the product family members and, on the other hand, explore possible ways that those members can vary.

6 CONCLUSIONS

A complete documentation of design decisions and their effects on the system under development is essential for the maintenance and evolution. Design decision trees and requirement spaces can be used to document this information and communicate it between designers and persons responsible for the maintenance of the system. Having a link from the requirements to the design and implementation increases maintainability and possibilities of reuse.

Requirement spaces can be also used when a developer creates a new system as a product family member. Variance visualisation helps product family development by assuring that the framework can support planned variance. This method allows design re-use between product family members and makes creating new member a much simpler task. In this way, we can achieve a structured and organised design, which is easy to maintain and change, while maximising re-use.

7 REFERENCES

Alonso, A., Leon, G., Duenas, J. and Puente, J. (1997) Framework for Documenting Desing Decisions in Product Families Development. Proceedings of 3rd IEEE International Conference on Engineering of Complex Computer Systems, Como, Italy 1997, 206-212

Ardis, M. and Weiss, D. (1997) Defining Families: The Commonality Analysis, in Proceedings of the Nineteenth International Conference on Software Engineering, May 1997, 649-650

Lane, T. (1990a) Studying Software Architecture Through Design Spaces and Rules. Technical Report CMU/SEI-90-TR-18, Carnegie Mellon University Software Engineering Institute, October 1990

Lane, T. (1990b) A Design Space and Design Rules for User Interface Software Architecture. Technical Report CMU/SEI-90-TR-22, Carnegie Mellon University Software Engineering Institute, November 1990

Karlsson, J. and Ryan, K. (1997) A Cost-Value Approach for Prioritizing Requirements. IEEE Software, IEEE, September-October 1997, 67-74.

Ran, A. and Kuusela, J. (1996) Design Decision Trees. Proceedings IWSSD-8, IEEE, 1996, 172-175.

Roberts, D. and Johnson, R. (1996) Evolve Frameworks into Domain-Specific Languages. *Pattern Languages of Program Design 2* (ed. Vlissides, J., Coplien, J. and Kerth, N.) Addison-Wesley, USA, 1996, 93-103.