The building block method. Component-based architectural design for large software-intensive product families
Müller, J.K.

Citation for published version (APA):

General rights
It is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), other than for strictly personal, individual use, unless the work is under an open content license (like Creative Commons).

Disclaimer/Complaints regulations
If you believe that digital publication of certain material infringes any of your rights or (privacy) interests, please let the Library know, stating your reasons. In case of a legitimate complaint, the Library will make the material inaccessible and/or remove it from the website. Please Ask the Library: http://uba.uva.nl/en/contact, or a letter to: Library of the University of Amsterdam, Secretariat, Singel 425, 1012 WP Amsterdam, The Netherlands. You will be contacted as soon as possible.
1 Introduction

1.1 Motivation

The motivation for the work presented in this thesis comes from current trends in the development of large software-intensive systems. Software has become an essential technology in the development of almost all complex systems. This is exemplified by the developments in the area of electronic products where software has become the flexible counterpart to chip technology. Functionality, which undergoes evolution and extension, is more and more implemented in software.

Families of products are conceived to cover a wide functionality spectrum using a common base. Product platforms are developed, with which extended or new functionality can later be offered to the customer in form of additional features. In general, the amount of software increases in complex systems.

To cope with these trends, software systems need to be increasingly modular and need to be built from reusable components. However, splitting functionality is only one side of the coin. The increased modularity makes system integration a pivotal step in the development of a system. Architecture plays a crucial role in handling modularity and achieving required system properties.

Current software development very often results in software systems which do not exhibit the kind of modularity which facilitates integration, evolution and extension. A frequent cause is that initially required functionality is taken as the sole basis to develop the architecture. Subsequently required features come as a surprise and are implemented in an ad hoc manner. Another cause is system integration not being planned for but entered unprepared resulting in long integration times where a lot of the global consistency has to be established after the fact. Furthermore, the architecture is not updated and maintained as necessary. The resulting lack of clear structure makes evolution and extension increasingly difficult.
Therefore, development of a product and its architecture needs to be supported systematically through appropriate methods. Current software design methods do not sufficiently address these issues.

Object-oriented design methods take the notion of an object as central point. This blurs the distinction between the modelling of an application domain with building a system. The notion of an object is quite different in these two areas. Application domain modelling captures the perception of a domain and uses an intuitive notion of an object to denote things. This led to the proposal of using the less restrictive notion of a feature instead of an object (see section 2.6.2). The notion of an object in building a system is quite different. An object encapsulates state and behaviour. Furthermore, the idea of a seamless transition between application domain modelling and system building ignores the quite different intention of the two tasks (in chapter 9 a more extensive analysis of other design methods is given).

Another problem is the search for a natural modelling of the software with respect to the application domain. This implicitly assumes that the domain functionality is the major part of the system functionality. In practise, this is often not the case. Take the example of the tss system (see appendix A) where the call switching functionality amounts to 20% of the total functionality while the rest of the functionality is in response to system qualities. This percentage may be higher for other systems but system quality induced functionality will always be a considerable part of a system’s functionality.

A third problem is that design methods do not take into account that for industrial products a commercial design activity takes place which determines the functionality of the products from a commercial perspective, identifying necessary and optional product features and their market priority. The priority of features of products is an important input for their timely development.

Finally, the advancement of SW component models makes the development of SW components technically feasible. However, methods are missing which yield the full power of component-based development: development of products from such components that the most likely product evolution and extensions can be done by changes to a small number of components only or by the development of a few new components.
1.2 Goals

The purpose of this work is to present a component-based architectural design method for the development of large software-intensive systems. The method, called the Building Block Method (BBM), focuses on the transition from domain-oriented decomposition to construction elements of a product family. A number of architectural models are developed to guide this transition. The name Building Block Method refers to its characteristic of supporting SW components, called Building Blocks (BB).

The BBM is designed to support the creation of product family architectures. Its main focus is identification and design of components including component frameworks and system infrastructure generics for the development of large-scale SW-intensive product families. An architectural skeleton of component frameworks and system infrastructure generics, which are stable for a whole product family, is one of the design goals of the BBM. Composing a product from pre-manufactured components is a way in which software reuse in a product family architecture can be achieved. The BBM addresses issues of configurability, in particular configuration to minimum; aspect design which complements object and concurrency design; factoring out into frameworks; extensibility; incremental integration and testing.

The BBM takes application domain functionality, commercial product features, system qualities and technology choices as input and produces a number of architectural models and construction elements. The main architectural model is an acyclic dependency graph of BBs to allow incremental integration and testing. Models for aspect designs, concurrency design and deployability design complement this model. The construction elements include the BBs, their various roles and their designs.

Architecture can only lay a good foundation on which the system is built. A good product needs more than a good architecture alone. Mistakes may be made in any of the various phases of system development: requirements, architecture, detailed design, implementation, deployment or documentation, to name the most important ones. An architectural design method needs to connect to the other phases. This connection is given in section 2.6 by the broader architecting context, which we assume for the BBM and in chapter 11 by the development process embedding.

We present the BBM in two steps:
a core method applicable to the design of large software system in general, and

a specialised method for designing centrally-controlled distributed embedded systems.

Experiences with parts of the method were gained in the development of systems in telecommunication [LM95b] and in medical imaging [Wij00]. One of the telecommunication product families, called Telecommunication Switching System (tss), from which the BBM has been bootstrapped, will also be used as an example of a concrete design made with this method. The product family is described in appendix A. Our experience comes from participation in the development of that product family. We developed design guidelines, redesigned the equipment maintenance subsystem and participated in its implementation (see section A.3.8 and section A.5.3). Since then, the BBM has been further consolidated and in parts be published. This thesis presents a systematic treatment of the method.

1.3 Contribution

This thesis makes several contributions to architectural design for large software-intensive systems. It introduces:

a component-based architectural design method for software-intensive product families,

aspect design which adds function design to object design (see chapter 5),

object design, aspect design and concurrency design as three design dimensions of architectural design (see section 3.3),

embedding of architectural design into an industry-proven rational architecting process (see section 2.6),

commercial product features to guide architectural design (see chapter 8), and

an incrementally integratable, extensible, component-based product family architecture (see section 8.3).

Variable functionality of a product family that goes beyond parameter values is encapsulated in SW components. SW components are the way to represent both common and variable parts of a product of a family. Aspect design deals with
functionality of large SW systems that is not derived directly from the application domain but is a consequence of quality requirements. It furthers conceptual integrity because aspect designs are orthogonal to object structures. The three design dimensions are key dimensions for designing functionality of large SW products. The freedom to assign functionality to SW components addresses the problem of the so-called tyranny of the dominant decomposition. This tyranny denotes the problem that the application of a certain development approach forces the same kind of decomposition upon all problems, for instance an object-oriented method will always lead to an object decomposition.

An explicit rational process for architecting relates the architectural design to other tasks which are necessary to develop SW products in industry. Commercial product features provide an important anchor for product modularity. The modularity induced by application domain objects is complemented by modularity induced by commercial features. Incremental integration based on layering of BBs is an important means to accelerate the integration and testing process of large software products.

Some of the material of this thesis has been published before. The concept of the composition of a product of a family from components has been published in [LM95a], integration of architectural design into the development process in [Mü195]. A general overview of the BBM was published in [LM95b]. How to relate features of a product with its software structure was published in [Mü197]. Aspect-oriented design as an additional concept to object-oriented design was elaborated in [Mü199].

1.4 Outline

In chapter 2 we will describe the context for the concepts introduced in this thesis. There we will define important terms such as system, architecture and method, describe a development context in which the BBM is intended to be used, give a historical background of the BBM, and sketch the BBM concepts and their relations within the method.

---

1. Besides the publicly accessible publications, a number of internal publications have been made. [LM95c] was the basis for the [LM95b], it contains examples which had to be left out because of space limitations. [LM95d] presents the BBM as a number of patterns. [LM97] explains the concept of virtual processors which enables budgets of processor time to be allocated to a group of processes.
With chapter 3 the main part of the thesis starts. It gives an overview of the BBM and shows how additional design tasks are related to the BBM.

The following three chapters explain these design dimensions in detail. Designing objects is dealt with in chapter 4. Chapter 5 introduces SW aspects as a functional structure orthogonal to the object structure. In chapter 6 the design of threads and processes is shown.

Chapter 7 explains the concepts that are used to define manageable and composable BBs. Furthermore, it describes how deployment sets are built. Chapter 8, then, introduces the concept of product family architecture. Its main point is that product features that are implemented in software are related to BBs.

Chapter 9 compares the BBM with other methods and approaches. The comparison will be based on the architectural meta-model of the various methods and approaches.

In chapter 10 a method specialisation for centrally controlled distributed embedded systems is presented.

Chapter 11 explains the consequences of the BBM for both the development organisation and the development process. One of the main points is the layered development process, that is, the architecture is developed in a process which guides the development process of BBs.

Chapter 12 discusses what has been achieved by the BBM and concludes the thesis.

Appendix A gives an introduction to the Telecommunication Switching System (tss) product family, which is used throughout the thesis as an example illustrating a BBM-based design.

Formatting

To stress the main train of thought potentially distracting details have been formatted as insets. This paragraph is an example of an inset. While such insets may contain interesting information for the curious and patient reader, they may safely be skipped when aiming at an understanding of the big picture.

Heuristic: Heuristics about the execution and application of the method are also presented as separately formatted paragraphs such as this one.