

2 Product Architecture: Key Concepts
and Implications

This chapter examines the first body of knowledge that is needed to explore the relationship between product architecture and organization. We will focus extensively on engineering design knowledge in order to gain understanding about product architecture and to explore how architecture can be clearly represented. In particular, there will be a focus on the precise definition and characteristics of the decisions that determine the architecture of a product. By elaborating these in great detail, a firm foundation will be laid for the later chapters of the thesis, where they will be placed in an organizational context.

To that end, several engineering design methodologies that have been proposed in the literature to guide engineers during the design process will be examined. These models contain extensive technical knowledge on how to construct a physical product and more or less explicitly contain very useful information about product architecture. However, a glance through the existing literature will be sufficient to show that there is no universally applicable engineering design approach. Instead, a great variety of distinct models with different terminology exists, making simple accumulation of knowledge a hard task. In this chapter a selection of engineering design methodologies will be made and the essential elements carefully described in order to enable valid interpretation of the constructs. Several methodologies will be discussed and compared in order to be able to choose a useful set of definitions. The following steps will be performed:

· An outline of problem solving will be presented in order to understand the general principles of engineering design methodologies.
· The key concepts of two distinct well-known design methodologies – VDI Design, and Axiomatic Design – will be provided, discussed and compared.
· Ulrich’s (1995) widely accepted definitions of product architecture will be described, and based the previous discussion of the engineering methodologies the definitions for this study will be chosen. Furthermore, how a particular architecture can be represented will be explored and discussed.
· The impact of product architecture on the manufacturing firm will be summarized in order to indicate what architectural decisions are broadly contingent upon.
· The various steps will be summed up.

2.1 Problem solving in general
Researchers in product development (and in particular in engineering science) conceptualize the design of a new product as a process in which the organization creates and defines problems and then tries to solve them (Alexander 1964, Simon 1981 Steward 1981, Nonaka 1994, Pahl & Beitz 1996, Thomke 1997, Smith & Eppinger 1997a)². Despite each problem-

² Note that publications of Simon related to this topic trace their roots in the early sixties and were one of the first with this point of view. However in this thesis is referred to ‘the Science of the Artificial’, published in 1981.
solving process being unique, it is commonly believed that all problem-solving processes share common characteristics. This section describes some basic principles of problem solving. This will be helpful for an understanding of the more specific engineering design methodologies, and will also be useful for the organizational theories in Chapter 3. Presently the following issues will be considered:

- Introduction to problem solving.
- Hierarchies in problem solving.
- Link with product development.

### 2.1.1 Problem-solving basics

Problem solvers are concerned with how things ought to be (Simon 1981). They try to realize desired situations. Take, for instance, the case of somebody being in Drachten but aiming to be in the city of Groningen. He or she takes action and catches a bus that brings him or her to the destination station. In problem-solving terminology, catching the bus is a description of a process that takes you from one state (being in Drachten) to another state (being in Groningen). Accordingly, a description of a future desired state will related to how the problem is presented (Simon 1981). In general the problem-solving task is to discover a sequence of processes that will realize the goal starting from the initial state. In order to find a solution, people generate alternative actions and test these against a whole range of criteria. For instance, when you have decided that you want to be in Groningen, there are many alternatives for taking you there. What happens in practice is that one generates alternatives (e.g. going by plane, car, train, bus, bike or on foot.) and these are subsequently tested against a whole range of criteria (e.g. total costs, traveling time, schedule, or availability). Finally, an appropriate alternative is selected and you implement your decision. The process of generating alternative solutions and subjecting them to a general test is referred to as generate-test cycles (Simon). The literature dealing with engineering design shows many small variations on this theme (Pahl & Beitz 1996, Blessing 1996) but broadly speaking they all agree upon the cyclic character of finding a solution. The concept of cycles will return later in this thesis (section 2.2, and Chapter 5).

According to Simon, problem solving is a matter of trial and error. People build up associations between particular states and specify actions that have to realize these changes. In general, the path towards finding a solution is not chosen blindly, but largely shaped by experience or rules to do with which actions should be tried first. As Simon points out:

*All that we have learned is that human problem solving involves nothing more than varying mixtures of trial and error and selectivity.*

In line with this, many researchers have examined the general rules of effective problem solving. These aim to propose methods that are helpful for finding a solution for a particular problem in an efficient manner. There is obviously a wide range and variety of approaches, though a concept that frequently appears in the context of effective problem solving is that of hierarchy (or decomposition). This will be discussed below.
2.1.2 Hierarchies

This section describes two hierarchical concepts that are suitable for complex problems. Each concept emphasizes different aspects and is often concurrently applied. The two are useful for the understanding of product development and in particular for product architecture. The following will be described:

- The vertically arranged layers involved in decision-making.
- The horizontally arranged sub-systems of Simon’s hierarchical concept.

Layers

Mesarovic (Mesarovic et al. 1970) states that in real problem-solving situations, people generally do not know the exact consequences of alternative actions. However, postponement of the decision as to what action to implement simply implies that no action is preferable. He illustrates the resulting fundamental problem-solving dilemma below:

*On the one hand, there is a need to act without delay, while on the other, there is an equally great need to understand the situation better.*

This dilemma can be solved if the solution to the original complex problem is substituted by the solutions for hierarchically arranged simpler sub-problems. That is to say, one defines a sequence of sub-problems for which the solution of a particular sub-problem completes the specification of the subsequent sub-problem that in turn can be attempted. The original goal is achieved when all sub-problems are solved. For instance, in order to reach Groningen you may first make the decision to go by bus, and then decide where and when you will start the journey (depending on the available buses). These decisions together determine how you will get to Groningen: the solution of the initial problem.

Mesarovic refers to such a hierarchy of decisions as a hierarchy of decision layers. Figure 2.1 shows a decision problem that is partitioned into k layers of sub-problems, where the output decision problem k is needed to specify decision problem k-1. Feedback between the decision units is plotted as dashed lines.

![Figure 2.1 Hierarchical layers according to Mesarovic](image-url)
Decisions at different layers generally have relatively different characteristics. Compared to a lower layer, a higher layer of decision making:

- is concerned with a larger portion or broader aspects of the overall problem.
- has a longer decision period.
- is concerned with slower aspects.
- is less structured, has more uncertainties, and is more difficult to formalize quantitatively.

It may be relevant to mention that the term sequence of sub-problems must not be confused with a necessarily sequential, top-down way of problem solving. A problem may be attempted by moving up but also by moving down the layers. Moving down the hierarchy is required since a downstream sub-problem cannot be exactly formulated without a solution being found for the upstream problems. On the other hand, moving from lower to higher layers is necessary since the solution of all sub-problems implies a solution being found for the overall problem. The success of a higher layer depends on the performance of the lower layers. If no solution is reached for a specific sub-problem this has to be fed back to a preceding layer or passed forward to a lower layer where it can be fed back to higher layers if necessary. While it is difficult to make firm statements about the decision-making sequence, this still illustrates the need for a multi-layer structure when dealing with complex problems.

**Simon’s hierarchy**

In addition to a hierarchy of layers that focuses on the vertical arrangement of problems, a problem may also be decomposed in a horizontal fashion. Let us return to the traveler. Suppose he has made up his mind and now decides he wants to visit Orchard Road in Singapore. This is a somewhat more difficult problem since there is an enormous range of possible actions that will take him to Singapore. To simplify the problem, he decomposes it into two smaller sub-problems: Groningen-Amsterdam and Amsterdam-Singapore. Each sub-problem can then be solved in relative independence of the other. However, the two sub-problems have to jointly generate a solution for the original problem and therefore have to have a degree of interaction. In the example, the arrival time in the first solution must be established before the departure time in the second sub-problem can be arranged. Furthermore, it is possible that both sub-solutions may together not exceed a certain budget or total travelling time. The decomposition decision is not unique. There may be many alternatives, and these may also lead to alternative processes and alternative solutions. An important feature, however, is the relative independence between the sub-systems. This brings us to the hierarchical concept postulated by Simon (1981).

A system (i.e. a problem) can be viewed as a unit as opposed to its environment. In real life a system is ‘open’ and has a relation with its environment, specified as system inputs and system outputs. A system can be described as a so-called black box. If that is the case, only its input and output are specified without considering its inner environment. In order to increase understanding of the details, a black box can be split up into smaller black boxes, each with inputs and outputs. In turn, these boxes can be decomposed again until the lowest hierarchical level is reached. This is depicted in Figure 2.2. As a result, a hierarchy is created consisting of an arrangement of interacting sub systems. It should be noted that Simon’s hierarchy not only states that a sub-system is part of a larger system (a volleyball player is part of a team, and the team is part of a whole club), but especially considers the interactions between the sub-systems (i.e. how the players interact).
According to Simon, a system is complex when it is made up of a large number of elements that have many interactions. Simon argues further that most systems are to a large extent decomposable. This means that a system can be decomposed into relatively independent sub-systems. Nearly decomposable systems can be split up into sub-systems such that the interactions within the sub-systems are much stronger than the interactions between the sub-systems. Hence an effective approach for dealing with complex problems is to decompose them into smaller subsystems that are each easier to handle, and are relatively independent.

These principles also apply to product development (Von Hippel 1990, Eppinger et al. 1994). Once a problem is decomposed, the interactions between the sub-problems should be well understood and managed. Furthermore, the amount of interaction influences the speed at which the overall problem can be solved. In fact, the number of interactions between the sub-systems determines to what extent the sub-problems can be solved concurrently and therefore strongly impacts on the speed of solving the original problem.

2.1.3 Link with product development
The above description of decomposition strategies is very summary but sufficient to provide a panoramic view of the most important concepts informing this thesis. General problem solving literature will therefore not be analyzed further, and instead a closer look will be taken at engineering design methodologies where the basics again occur but now in a more specific form.

- Before doing so, an indication will be given of where the main lines occur later.
- Engineering methodologies distinguish the functions of a product from physical solutions that have to achieve the functions. This is similar to the distinction between a desired state or goal and the process that takes you to the desired state.
- In order to find a solution relating to a product’s function, engineering design methodologies prescribe conducting generate-test cycles.
- Engineering methodologies prescribe a whole sequence of stages and steps that need to be taken to design a product. This is fully in line with the layer structure of Mesarovic.
- Within engineering models, problems are constantly being decomposed in smaller interacting sub-problems. Simon’s hierarchy applies in particular to product architecture, which deals with how a product can be decomposed into physical building blocks (physically distinct units or chunks of a product) and how these blocks interact.

These aspects will be explained in the following sections.
2.2 Design methods
This section will explore a large set of different engineering design methodologies. The overall model area will first be described and the prescriptive design literature then examined. The design methodology of Pahl and Beitz (Pahl & Beitz 1996) will be looked at, as well as the Axiomatic design approach taken by Suh (1990). Finally, the two approaches will be compared and discussed, especially in relation to the definitions and interpretation of the technical constructs that are used.

2.2.1 Introduction to engineering design models
A large number of models and methodologies have been developed within the field of engineering science. These ultimately aim to direct decisions and activities during the design process in order to improve performance. To some extent they all include the basic characteristics of problem solving described in the previous section. Not surprisingly though, engineering science offers a great variety of different solutions (Malmqvist 1995, Malmqvist et al. 1996). In general these are divided into two broad categories: describing and prescribing methods (Blessing 1996, Erens 1996, Stake 1999).

Describing models addresses how design processes actually take place. This category can be further divided into cognitive and other studies at an individual level and studies at a group level. The latter may consist of models that illustrate the problem-solving structure of design teams (see the Design Structure Matrix models in Chapter 3.2) or include more general comparative case studies of problem-solving strategies and related performance.

Prescribing methods consist of a collection of formulas that prescribe a rational and structured way of working that purport to lead to an effective and efficient design process. These methods are generally based on the personal experiences of the authors. This category consists of two streams. In the first place, prescriptive methodologies that prescribe a particular process (course of action) that is necessary to bring a product to its final shape. Secondly, artifact models that focus on the outcomes of the process. These describe the evolving states of the product. Quality Function Deployment (QFD) and Axiomatic design are well-known methods within this context. QFD describes a product from the client, design, component, and manufacturing perspectives, and links these points of view (Hauser & Clausing 1988). Axiomatic design addresses particular states of the product by means of modeling and analyzing the evolving decompositions of the original problem (Suh 1990).

Figure 2.3 shows the division of models described above inspired by existing categorizations (Blessing 1996, Erens 1996, Stake 1999). The classification must not be seen as exclusive, but rather as highlighting some relative differences. Putting prescriptive literature and descriptive literature into opposing camps would be too rigid an approach. There are some good reasons why these have at least some overlap. In fact, the prescriptive models originate from the personal experience of designers in the field and thus are implicitly based on practice. Furthermore, it is reasonable to expect that the teaching of prescribing models to engineers will have affected their way of working. Moreover, scientifically speaking, one would expect that both categories to have become highly interwoven over the years. Scholars should explicitly test and adjust prescribing models based on descriptive models and the other way around. However, the latter argument can only have limited validity since prescriptive models have rarely taken descriptive models into account (Blessing 1996).
In any case, it is reasonable to assume that the basic elements of prescriptive literature are valid and can be applied in practical situations. This is important for the purposes of this thesis since the prescriptive models will be pursued here with the aim of the insights being applicable in practice. The rationale for focusing on prescriptive models here is that these include a precise definition of technical constructs (functions and technical solutions) that are needed to define product architecture in the next section. However, there is more at stake than definition alone. The aim is also to explore the ways that technical constructs such as functions and technical solutions affect the course of the design process of a product according to the prescriptive models. This insight is helpful in enabling a proper interpretation of the characteristics of a product architecture and placing them in an organizational context later in this thesis.

As indicated previously, there is no universally valid design methodology and accordingly no clear uniform definition of the technical terminology. Both prescriptive methodologies and axiomatic design contain very useful insights for defining and interpreting product architecture.

Below the two will be described separately such that their underlying paradigms are made quite clear. The two models will then be compared and described and the foundation laid for the research terminology. The prescriptive methodologies will be described first, followed by axiomatic design.

2.2.2 Prescriptive design models
Prescriptive design methodologies are usually very detailed handbooks describing many stages, activities, and examples, and containing a large amount of technical knowledge. In general, prescriptive design methods divide design processes into stages. Each stage includes the process that takes place between two states of a product. Performing the processes belonging to all stages brings one from the initial state (idea) to the final state (full specification). Furthermore, prescriptive methodologies actively guide human problem-solving
activities such as identifying the problem, generating solution alternatives, selection of the best one, and implementation.

Again, there is a great variety of methodologies (Tate & Nordlund 1995, Andreasen et al. 1996, Erens 1996), each with different steps, definitions, foci and different optimal paths for problem solving. Basically, however, the prescriptive methodologies have many common features. Based on an extensive literature study, Blessing (Blessing 1996) concluded that all methodologies roughly fit within a division of three stages: a problem definition stage, a conceptual design stage, and a detail design stage. Each of these stages addresses the process by which a particular state of the evolving product is reached as shown in Figure 2.4.

All methodologies draw a distinction between the function of a product (what it has to do) and the physical solution (how it is achieved). They usually translate customer requirements into functions, and try to find technical solutions that fulfill these functions such that in the end a fully specified product results.

In order to further elaborate these issues, the VDI design will be focused on, since this is generally representative of a large body of engineering models. VDI was described by Pahl and Beitz (Pahl & Beitz 1996), and is well known in many engineering areas. In fact, VDI is commonly known as 'Pahl and Beitz'.

**VDI Design**
The Society of German Engineers (VDI) devised a methodology (VDI 2221) described in great detail in “Engineering Design: a systematic approach” by Pahl and Beitz. The authors describe four stages including a number of steps guiding the design of a product from scratch to full specification, as illustrated in Figure 2.5. The stages are planning and clarification of the task, conceptual design, embodiment design, and detail design. These will each be described under the headings below. There will be a particular focus on the conceptual design phase where the concepts of functions and working principles are defined in detail.
Figure 2.5 Steps in a design process according to Pahl and Beitz
Planning and clarification of the task

This stage starts with an incentive: bringing a new product to the market that has to be attractive in terms of market conditions and company strategy, and concludes with a list of requirements that need to be fulfilled by the new product. To that end the company will conduct an extensive analysis of the marketplace and situation of the firm and subsequently set a process in motion where product ideas are generated and selected. The most promising idea is then refined by formulating a product proposal that clarifies the product’s task. Finally, the company creates a list of product requirements that in turn is used to set the next stage of the design in motion.

Conceptual design

The conceptual design stage produces the principle solution required to establish the product requirements. The stage begins with an analysis of the main problem that needs to be solved to satisfy the list of requirements created in the previous stage. The following steps are then taken:

· Construction of the function structure.
· Searching for and selecting working principles.
· Combining the principles into a working structure.

These steps determine the main product structure and will be paid special attention. A designer first needs to formulate an overall product function. A function describes the relationship between inputs and outputs within a system. These inputs and outputs can be categorized into three types: flows of energy, flows of material, and flows of signals (information). A function thus expresses a transformation of energy, material, or information. Functions are preferably described as a verb-noun pair without a preconceived idea of the solution. For instance, ‘decrease temperature’ may be a function of a refrigerator. This description does not include any indication of how a solution to lowering the temperature can be found.

When the overall function is clearly specified, the design evolves to the stage of decomposing the overall function into smaller functions. These functions are again transformations of energy, material, and information, but at a lower level of complexity. The resulting set of functions can be arranged in a function structure, such as is shown in Figure 2.6. The structure indicates that all functions are part of the overall function and can be connected to each other. The output of the one function becomes the input of the other function. All of the connected flows together constitute the input and output of the overall function.

In addition, Pahl and Beitz classify functions as being main or auxiliary (as can be seen in the function structure depicted in Figure 2.6). They state that the main functions contribute to the overall function directly, whereas auxiliary functions have a more supportive character and contribute to it indirectly. For example ‘decrease temperature’ is probably a main function of a refrigerator, but may be considered as auxiliary for a personal computer. Hence a computer is not designed to decrease temperature, but its temperature must be lowered to prevent the processor from overheating. The distinction between the main and auxiliary functions affects the prescribed sequence of problem solving. It is advisable to start with the design of the flows of the main functions and afterwards address the auxiliary flows.
Second, once the functions have been specified, the search for appropriate solutions can start. The final solution for the overall function is obviously not directly available and has to be created step by step, piece by piece. Hence, the role of the function structure is to guide the search for solutions. It enables problem decomposition and facilitates the recognition of parts for which the solutions are known or available. The level at which the decomposition has to take place depends on the level at which the search for solutions for each sub function seems most promising. When existing physical solutions can be assigned directly, the decomposition may end at a relatively high level. For totally new design, the decomposition has to be performed until levels of much lower complexity are reached.

Third, once the functions are clearly specified, the search for solutions can be dealt with concurrently. A working principle has to be chosen for each function. A working principle expresses basic physical characteristics (geometry or material) to realize a physical effect that is needed to perform a given function. For instance, a working principle may be depicted as a rough sketch of a leverage that is based on the leverage law (physical effect) realizing the function ‘amplify force’. Mapping each function in order to develop a working principle may be guided by a morphological scheme. For each function, collection of several alternative working principles is considered, from which the most appropriate one is selected.

Once a working principle has been chosen for each function, the challenge is to combine these working principles such that they together fulfill the overall function of the product. The combination of working principles is primarily based on the input-output relationship established clearly in the function structure. That is to say, each working principle has to realize its corresponding functional inputs and outputs. However, this is generally not sufficient. The compatibility of working principles is often strongly affected by physical and geometrical considerations. Alternative combinations of working principles have different effects on technical and economic criteria. Making a selection of physically feasible, and technically and economically favorable combinations is generally a hard task for designers. Taken together, the choice and combination of working principles results in a specification of an overall solution principle that is the starting point for the next stage.
**Embodiment design**

Designers will now proceed with the determination of the overall layout (construction structure) in line with technical and economical criteria. The physical parts are roughly arranged and the forms (shape and material) of the components determined. Designers have to check for function, strength, spatial compatibility, and financial viability. In addition, factors such as safety, ergonomics, production, assembly, operation, logistics, maintenance, recycling, and costs are considered. In dealing with all these aspects, designers will discover a great number of interrelationships, making iteration unavoidable. Pahl and Beitz aim to aid designers at this stage by providing a great number of checklists and guidelines. It is striking that they spend 74% of the pages in their book on describing embodiment design.

**Detail design**

During this phase, the arrangements, forms, dimensions, and surface properties are laid down in their final form. The materials are specified, production possibilities assessed, costs calculated, and all drawings and other documents are produced. Detail design has a major impact on production costs and quality, and therefore on market success.

The four stages outlined by Pahl and Beitz have now been described in sequential order. These stages may be regarded like the four runners in a relay race. Once the first stage is finished, the baton is passed to the next stage that in turn provides the trigger for the subsequent stage, and so on. It should be noted, however, that the metaphor of a relay race is only valid to a limited extent. In fact, Pahl and Beitz stress the iterative nature of problem solving.

In the first place, each stage includes steps where alternative actions are generated, tested, and selected. These cycles are each described within the specific context of the stage in which they should occur.

In the second place, cycles between the stages are also part of the potential of the problem-solving path such as is shown by the dashed lines in the illustration of the stages. Pahl and Beitz recognize and accept that these cycles may occur, but do not explore the cycles between the stages any further.

In the next section, the principles of axiomatic design will be described. Subsequently, VDI design and the axiomatic approach will be discussed and compared.

### 2.2.3 Axiomatic design

Suh 1990, Albano & Suh 1992, and Albano et al. 1993 provide a general framework for structuring a design process called axiomatic design (AD). The key aspect is that designers are able to understand what the objectives of a product are, and the means by which these objectives are achieved. This framework is based on a fundamental set of design principles that (according to Suh) determine good practice. The axiomatic design group at MIT strongly advocates these principles and has devised many applications. Outside this group, axiomatic design is well known, but applied far less frequently. The abstract theoretical concepts require a lot of training and practice, and the strict rules probably constitute a stumbling block for many researchers. Nevertheless, many engineers refer to the basic principles and make use of various other aspects. The key elements are:
• **Domains**: The design process is modeled as the processing of information between the functional and the physical domain.

• **Hierarchy**: The design process progresses from a system level to more detailed levels. Decisions about the artifact are structured within a hierarchy in both domains.

• **Zigzagging**: The decomposition of the problem follows a top-down approach between the hierarchies of the two domains.

• **Design axioms**: These provide a basis for evaluating the design structure in order to realize good design quality and a smooth working sequence.

In general, axiomatic design develops products by continuously describing the functions and solutions within a set of constraints (Tate 1999). When the design process is initiated, the functions and solutions are described in a very abstract and simple manner. As the process evolves the product is illustrated in increasing levels of detail and by the time the product is finished every physical detail is known exactly.

The unique feature of axiomatic design is that it provides a framework that shows in detail the interplay between the functions and solutions at every moment of the design process. More precisely, axiomatic design works with a system of functional requirements, physical design parameters and constraints. These will be defined below.

Functional requirements (FRs) are described as elements in the functional domain and represent design goals. Such requirements state what one wants to achieve and should be stated as a verb-noun pair without noting a particular solution. FRs can be decomposed at several hierarchical levels (see Figure 2.6), but at each specific level FRs are by definition independent of each other.

Physical design parameters (DPs) are elements of the physical domain and have the purpose of satisfying a particular FR. DPs describe how the FRs are achieved. That is to say, for each FR a DP needs to be selected in order to achieve the FR. The DPs can also be taken down to lower levels of abstraction. It is striking that the exact meaning of a physical DP depends on the hierarchical level in which it is considered. At the lowest hierarchical level, DPs become physical parts or very precise specifications of geometry, material, or tolerances (Hintersteiner & Friedman 1999). At higher levels, though, DPs are not necessarily physical and may represent general solution principles or concepts. The term ‘physical’ before the DP does not therefore imply that every DP is a piece or subassembly of a physical product (Albano & Suh 1992). A hierarchy within the physical domain is not by definition a hierarchy of a physical product describing the whole product, its building blocks, its subassemblies, its parts, etc. The term ‘solution domain’ would probably have been a much clearer indication that only the lowest level of the DPs refers to physical things that can be located somewhere on a physical product.

The constraints are specifications that the design solution must possess in order to be acceptable in the eyes of the customer and the designing or producing firm. In general, constraints depend on many decisions. Constraints impact on (multiple) FRs and limit the range of feasible DPs (Hintersteiner 1999). Recent work on AD (Tate 1999) has identified many different types of constraints. For the purpose of this thesis, however, the so-called global constraints that affect many design parameters and cannot be allocated to a particular function or solution (Albano et al. 1993) are the main ones under consideration. Examples of
global constraints include restrictions in weight, size, or costs. The weight of the overall product for instance, is clearly a result of all components together and cannot be allocated to only one feature.

Having described the above constructs, the prescribed decomposition within axiomatic design will now be examined. Figure 2.7 shows that if FRs and DPs are decomposed, the graphic effect is a zigzag pattern. At the highest level of abstraction there is just one FR that needs to be satisfied by just one DP. The selection of an appropriate DP has the characteristics of a generate-test cycle (Albano & Suh 1992, Tate & Nordlund 1995, Tate 1999). Multiple DPs are generated and tested until a satisfactory one has been selected. Only if the FR has a corresponding DP can it be decomposed into a set of sub-FRs. At this hierarchical level a sub-DP needs to be selected for each FR in a similar fashion to that just described. Again, once all sub-FRs are linked with a sub-DP, each sub-FR can be decomposed and the whole procedure repeats itself until the lowest level is reached (see e.g. Tate 1999).

![Figure 2.7](image)

Axiomatic design pays considerable attention to the interplay between the FRs and the DPs. The so-called Axiomatic design matrix (A) indicates how the DPs together address the FRs at each level of the hierarchy. This is shown by the following equation: \( \{FR\} = [A]\{DP\} \) and illustrated by the three diagrams below. The structure of the axiomatic design matrix determines the sequence in which the product is designed. The matrix distinguishes between three different forms. It can be uncoupled (Figure 2.8), de-coupled (Figure 2.9) or coupled (Figure 2.10).

![Figure 2.8](image)

A design is uncoupled when only the diagonal elements (i=j) of the matrix have an ‘X’ and all others (i ≠j) have an ‘O’. In that case each DP only impacts on its own FR and the DPs can be altered completely concurrently until each one is able to establish its FR. It should be noted

24
that within the matrix all diagonal elements have by definition an ‘X’, since for each FR a corresponding DP has been chosen.

In practice, however, it is very likely that there are DPs that affect more than one FR. As a result the axiomatic design matrix also has non-diagonal elements filled with ‘X’s. A design is de-coupled when the matrix can be written in a lower triangular form, as can be seen in Figure 2.9. This implies that there is at least one DP that impacts on multiple DPs, and fully independent adjustment of DPs is not a realistic option anymore. Instead, the DPs need to be altered in a sequential fashion in order to achieve the FRs exactly. For the example shown in Figure 2.9, this means that first FR21 needs to be satisfied by DP21. Next, FR22 will be achieved by DP22 in collaboration with the already specified DP21. For FR23 it is the same story, but DP23 has to complete the effects of DP21 and DP22. Note that any other working sequence will result in unnecessary iterations.

\[
\begin{bmatrix}
FR21 \\
FR22 \\
FR23 \\
\end{bmatrix}
= 
\begin{bmatrix}
X & 0 & 0 \\
X & X & 0 \\
X & X & X \\
\end{bmatrix}
\begin{bmatrix}
DP21 \\
DP22 \\
DP23 \\
\end{bmatrix}
\]

\textbf{Figure 2.9} A de-coupled axiomatic design matrix

Finally, a design is coupled when there are also relationships in the upper diagonal part of the matrix. In the example given in Figure 2.10, it can be observed that both FR21 and FR22 are uniquely fulfilled by DP21 and DP22. When the process starts by finding a solution for FR21, this solution needs to be altered in the case of FR2. When the designers start defining the right setting for DP21 and DP22 in order to achieve FR22, this in turn affects the performance of FR21. Quite clearly the resulting design process is highly iterative, and there is generally a long way to go before a setting for the DPs is found that is satisfactory for both F21 and F22. In Suh’s opinion, a coupled design is an example of ‘bad design’ and should be avoided.

\[
\begin{bmatrix}
FR21 \\
FR22 \\
FR23 \\
\end{bmatrix}
= 
\begin{bmatrix}
X & X & 0 \\
X & X & 0 \\
X & X & X \\
\end{bmatrix}
\begin{bmatrix}
DP21 \\
DP22 \\
DP23 \\
\end{bmatrix}
\]

\textbf{Figure 2.10} A coupled axiomatic design matrix

To sum up, the basics of axiomatic design have now been outlined. These will be helpful for the definition of product architecture but can also be used to analyze the product architecture in the later chapters of this thesis.

In the next section, the definitions of Pahl and Beitz, and axiomatic design will be discussed and compared in order to create a sound basis for defining and interpreting product architecture.
2.2.4 Discussion

This section compares the two engineering approaches described above. This is done for two main reasons.

The first reason is general scientific interest. It is very tempting to describe and compare two well-known methodologies that on the one hand have a similar goal of guiding the design process, but on the other hardly have any other aspects in common. Both have interesting and useful features and perhaps a thorough discussion of both paradigms will have future relevance. It is to be hoped that a detailed and precise discussion of their underlying rationale will prevent an indiscriminate combination of principles without any understanding of the exact differences between the two concepts.

The second reason refers much more directly to the central role of product architecture in this thesis. The definition of product architecture will be presented in the next section and will be based on a definition of functions and physical elements. In order to understand these definitions unambiguously and make further interpretation possible, the precise background(s) of these constructs needs to be described. The definitions of the two methods will thus be considered synonymously, and comments about their differences made. After the main constructs of product architecture have been presented in the next section, the discussion here will be revived and a choice made about the definitions that will be used during the remainder of this research. Having good definitions and, moreover, knowing what prescriptive approach can be referred to in order to analyze the product architecture, will provide a sound basis for the research.

At a general level, VDI provides an illustrative overview of all steps that need to be taken during a design process. These steps are easy to understand and suggest effective ways of working towards a final solution. However, VDI does not really focus on how decisions need to be made and what the potential consequences of particular decisions are. That is to say, while the whole approach is largely based upon the principles of decomposition, it does not indicate what ‘good’ or ‘bad’ decompositions are in a specific situation.

On the other hand, AD provides a clear focus on the underlying decomposition of a product and continuously indicates the effects of a particular design decision on the overall structure of the design process. However, the axiomatic design approach is very conceptual and surrounded by strict rules. A well-trained eye is needed to fully envisage a product within the framework of multiple levels of abstraction.

Compared with AD, VDI does not explicitly consider the functional and physical domains or the mapping characteristics they have in common. At first glance, though, the function structure is similar to the functional domain, and the concepts of working principle and working structure come close to the physical domain (Tate 1999). Furthermore, both approaches describe the role of generate-test cycles in finding a solution for a specific function. The following differences will be discussed:

- The definition of a function.
- Types of functions.
- Decomposition of a function.
- The combination of physical solutions.
The definition of functions

VDI and AD have in common that a function has to be specified as a verb-noun pair, preferably in a solution-neutral manner. However, AD has a much broader definition of functions than VDI.

AD defines functions as a design goal, or what needs to be achieved. VDI does not contradict this but has a more specific (narrow) definition. VDI defines functions as the transformation of material, energy or information. This implies that important design goals such as aesthetics cannot be captured within the function structure of VDI since it is barely possible to describe them in input-output language. As a result, important design goals are not captured within a VDI function structure at an early stage of design, but come in much later in the design process.

The point is that while every VDI function can be considered a goal (AD), not every goal can be described as a transformation of energy, material, or information.

This research suggests that it would be wise for VDI to take important design goals into account as early as possible in the design process and not to wait, because these cannot be illustrated as a transformation of material, energy, or information.

Different types of functions

VDI draws a distinction between main functions and auxiliary functions whereas AD considers all functions as essentially the same. The VDI distinction is based on the question of whether a function makes a direct or indirect contribution to the overall function. However, why Pahl and Beitz label one function differently than another is a question that perhaps needs to be asked. The answer is probably not that one is more important than the other since both types are eventually needed to fulfill the overall function. To return to the previous example, it is quite clear that if the PC cooling system doesn’t function, the computer will soon stop working. Hence, both the auxiliary and the ‘main’ functions are essential. The only reason that can be suggested is that the difference is based on a preconceived sequence of working, as the following would suggest:

Auxiliary functions have a supportive or complementary character and are determined by the nature of the solution. (p.32) It is useful to start with determining the main flows in a technical system. The auxiliary flows should be considered later. (p.156) The auxiliary flows help in the further elaboration of the design in coping with faults, and in dealing with problems... (p.156) The complete function structure, comprising all flows, can be obtained by iteration, ...first...the main flow...completing that by the auxiliary flow...and then establishing the overall structure. (p.156)

Alternatively, within AD the sequence of working is based on the structure of the AD matrix, or based upon the top-down principles of the problem-solving hierarchy. Put differently, the design process generally starts with a parent FR and once the solution is found it proceeds with child FRs (Hintersteiner 1999). Within the same hierarchical level, the AD matrix determines the (optimal) sequence of solving the FRs.

Taking this into account, it is at the very least remarkable that auxiliary and main functions are considered with same degree of detail in view of the fact that the cited passages state that the auxiliary function is fully specified by the solution of the main function. This apparently suggests that auxiliary functions may be considered on a lower hierarchical level than the
main function (according to zigzagging. This will be discussed below). There would thus appear to be no real reason for Pahl and Beitz to have identified two types of function (see also Tate 1999).

Decomposition
AD and VDI differ theoretically in how they perform a decomposition. AD prefers the zigzagging approach and VDI first makes a fully functional decomposition that one assumes is solution-neutral. Only when all sub-functions are completely specified does VDI initiate the search for sub-solutions. Subsequently these sub-solutions are abstracted to the whole again. Figure 2.11 outlines how the various decomposition sequences operate. For simplicity, only FRs and DPs are plotted.

![Figure 2.11 The decomposition strategies of AD and VDI compared](image)

This shows that Pahl and Beitz decompose in an U-shape, first a full decomposition of the overall function and then a bottom-up approach to reach the final solution. On the other hand, AD communicates between the functional and physical domains at each level of the hierarchy.

Having made this difference clear, it should be noted that Pahl and Beitz deviate considerably from their prescribed decomposing sequence in their examples. Among other things, they make the following remark:

*It should be remembered that function structures are seldom completely free of physical or formal pre-assumptions. Hence, it is perfectly legitimate to conceive a preliminary solution and then abstract this by developing and completing the function structure by a process of iteration.* (p. 160)

In conclusion, it is reasonable to assume that a function cannot be decomposed without considering its solution (on that hierarchical level). This is a very important concept and one that will be used again in Chapters 4 and 5.

Combining physical solutions
VDI and AD both aim to search for a solution for each function (at a particular level). However, the way that all of the solutions together fulfill the overall function is modeled differently.
The advantage of VDI is that its functional scheme provides a clear overview of the inputs and outputs that each working principle has to realize in order to obtain correct functioning of the whole. The DPs within AD also obviously have to realize the correct specifications of the FRs, but how all these flows are connected is not clearly visible.

The drawback of VDI is that it does not model interdependencies between working principles due to reasons other than functional inputs and outputs. A working principle that affects more than one function at a time, for instance, cannot be captured within the VDI function structure. Pahl and Beitz argue that technical and economic reasons affect the arrangement of working principles, but do not formalize these nor give any indication of how to deal with these interactions.

The advantage of AD is that it formally models how a set of FRs relate to a set of DPs and what the implications are. This is clearly visible when a DP affects multiple FRs. There is an ongoing focus on 'good' or 'bad' decompositions as early as possible in the design stage. The following can thus be concluded:

- AD defines functions as design goals and is more general than a transformation of energy, material, or information (the VDI function). However, the function concept of VDI may be considered a sub-set of the AD definition of functions.
- Pahl and Beitz distinguish between main and auxiliary functions but no convincing reason for the necessity of this division can be detected.
- A close look at both approaches indicates convincingly that a function cannot be decomposed without addressing its solution.
- The VDI functional scheme provides a clear overview of how the inputs and outputs interact, and AD shows how the mapping between functions and solutions is established. Both elements are important.

The next section explores the definition of product architecture. This will be followed by a return to the above discussion and a set of definitions that will be used during the remaining research will be devised.

2.3 Product architecture

Having developed a working idea of the main decisions that take place during a design process and having discussed the basic technical constructs (functions and solutions), the time has come to define product architecture. In broad terms, product architecture can be defined as the way that distinct product building blocks interact in order to obtain correct functioning of the whole. Section 2.4 will illustrate that product architecture has an enormous impact on the manufacturing firm, and is deeply embedded in a company's way of working. The present section will initially address the definition of product architecture and its characteristics in considerable technical detail based on the work of Ulrich (1995).

First, the definition of product architecture will be defined and the three technical decisions that determine product architecture explored. Ulrich's views on defining functions and solutions will be described, and the discussion will conclude with a clear set of definitions that will be used during the remainder of the research.

Second, product modularity, an essential characteristic of product architecture, will be examined.
Third, the means available to represent a particular product architecture in a fashion such that it can be analyzed will be discussed with reference to the work of Pimmler and Eppinger (Pimmler & Eppinger 1994) who model the interactions between building blocks.

Even though this section is relatively small and rather technical, it is essential for the study as a whole. Not only the technical decisions relating to architecture but also the modeling of interactions will form the basis for a linking of product architecture to organization in Chapters 4 and 5.

2.3.1 Definition of product architecture and modularity

In the foregoing sections it was observed that there is a myriad of research methods and terminology within engineering design literature. Definitions concerning product architecture form no exception to this (Pahl & Beitz 1996, Erixon 1998, Stake 1999, Blackenfelt 2000). However, instead of analyzing the entire set of alternative definitions of product architecture, the basic assumptions and definitions of Ulrich (1995) will be largely adopted. He managed to combine elements from several schools and proposed a technical definition of product architecture that is widely accepted in the literature.

In order to define product architecture, Ulrich basically considered the functions and the physical building blocks of a product. He proposed three technical decisions that together form the architecture of a product. These are:
- The arrangement of functions.
- The mapping from functions to physical building blocks.
- The specification of the interfaces among interacting physical building blocks.

![Figure 2.12 A conceptual illustration of product architecture](image_url)

The definitions of functions and building blocks

Before these decisions can be further described, the constructs of functions and physical building blocks according to Ulrich must first be defined. They can then be integrated into the previous discussion.

Ulrich states that a function is what a product does as opposed to its physical characteristics. Functions can be arranged by linking their inputs and outputs (of energy, material, and information). In addition, Ulrich recognizes that not all functions can be included within an input-output language. Ulrich and Eppinger 2000 recently stated the following:

> Also note that in some applications the material energy, and signal flows are difficult to identify. In these cases, a simple list of sub-functions of the product without connections between them is sufficient.
Ulrich defines physical building blocks as physically distinct chunks of a product. The building blocks of a computer, for instance, are the monitor, keyboard, hard-disk, etc. A product consists of building blocks that are in turn each made up of physical elements or components. Physical elements include the keys, the board, the switches, the LEDs, etc. These elements include the characteristics that are required to fulfill the functions.

People often talk about modules instead of building blocks. However, that is generally avoided within the literature since this has an implicit association with modular and not every building block is modular.

Based on the previous discussion in section 2.2.4 the constructs that will be used during the remainder of this research will be outlined below.

- A **function** is a design goal, describing what needs to be achieved (in line with AD). If it is possible, functions are conceptualized as transformations of energy, material, or information (in line with VDI). No distinction is made between main and auxiliary functions.
- **Building blocks** are purely physical and are not the same as the physical domain within AD, or working principles. However, it can be said that building blocks implement working principles or locate design parameters. Since at the lowest level of the physical domain DPs become physical things, it can be stated that each building block locates a whole collection of DPs (at the lowest level of abstraction) that ultimately fulfill the overall function of the building block.

Together these definitions allow an interpretation of product architecture based on the detailed considerations of the previous chapters. All the definitions are in line with axiomatic design, and the clarity of Pahl and Beitz has been added where possible. Ulrich’s three architectural decisions will now be described.

**The arrangement of functions**

Figure 2.12 shows an arrangement of a product’s functions. It shows that functions A, B, C, and D are connected by flows of energy, material, or information. Function F is a design goal that cannot be captured by input-output language.

**Mapping from functions to building blocks**

The bold boxes in the same diagram indicate the physical building blocks of a product. These blocks each implement the functions. Block 1 implements function A, block 3 achieves functions C, and D, and function F is established by blocks 4 and 2 together. An important characteristic of architecture is thus its mapping, by means of which building blocks are connected to a function or functions.

Possible mappings between functions and building blocks are one-to-one, many-to-one, and one-to-many such as is depicted in Figure 2.13 (where M=N=2). A combination of one-to-many, and many-to-one results in a many-to-many mapping (M-to-N). Note that for clarity the functions and the building blocks refer to the same level of abstraction.
Physical interfaces and coupling

Physical interfaces between interacting building blocks enable the physical realization of exchanges of energy, material, and information, and/or their geometric connection. A keyboard and a personal computer, for instance, are connected by a cable that realizes the exchange of information and energy. Two blocks can have a physical interface even when they do not exchange any flow. A plastic bottle and its cap have an interface but no exchange of material, information, or energy (except if one decides to model all forces between the two blocks, which is not common practice).

Physical interfaces cause a certain coupling between the blocks. The geometry of the cap, for instance, cannot be endlessly adjusted without changing the geometry of the bottle. Ulrich presents the following definition:

An interface is coupled if a useful change to one building block affects a change to the other block in order for the overall system to work correctly.

Note that this definition may refer to anything. However, a closer reading of the work of Ulrich suggests that the coupling refers to ‘physical’ aspects. Based on the examples in the paper, three reasons for coupling can be identified:

- A geometric connection that hampers the geometric freedom of both blocks.
- Side effects such as vibration, heat or magnetism of one block that effect the other block.
- Limited space that limits the freedom of size of both blocks.

Within product architecture the amount of coupling is an important feature. Interfaces may be coupled or ‘de-coupled’. Coupling is obviously a relative property. It is only possible to say that one interface is more coupled with respect to a particular change than another interface. For example, a mouse that is connected by a cable to a personal computer is more coupled then a cordless mouse since the latter is much less limited with respect to the position between the mouse and the computer.

Furthermore, it is worth emphasizing that coupling depends on what is considered a useful change. If it were feasible to control a computer with a cordless mouse at a distance of 100 kilometers then the cordless mouse would have a coupled interface. As things stand at the moment, however, this is not relevant.

To sum up, coupling refers to anything that constrains the design of another building block, though it is advisable to only consider relevant constraints.
Modular versus integral

Perhaps the most important property of product architecture is the dependence between its building blocks. To reduce things to the simplest terms, there are two types of architecture: modular and integral.

When building blocks are completely independent of each other the product is ‘modular’. On the other hand, a strong dependence between the blocks corresponds to an ‘integral’ architecture. In the most radical situation, a completely integral architecture will in fact include a product that cannot be decomposed into building blocks. All its pieces are equally strongly dependent. An ultimately modular product would actually consist of a collection of fully separate blocks that form no whole. In practice, however, there are hardly any such extreme cases and modularity is a relative property. For instance, a standard personal computer is more modular than a hand-held computer, and an SLR camera is more modular than an instant camera (for examples, see Ulrich & Eppinger 2000).

Ulrich defines modularity very precisely based on the previous described features: mapping and interface coupling.

- A modular product has a one-to-one mapping between functions and building blocks, and its interfaces are de-coupled.
- An integral product has complex (N-to-M) mappings between the functions and the building blocks, and its interfaces are coupled.

In general, products are neither entirely modular nor integral and lie somewhere between the two extremes. These are called hybrid architectures, and they have few non one-to-one mappings and some coupled interfaces. As a rule, the more the mapping is one-to-one and the more the interfaces are de-coupled, the more modular a particular product architecture is. This is illustrated in Figure 2.14.

![Figure 2.14 The range between modular and integral architectures](image-url)
2.3.2 Representing Architecture

Having addressed the basic architectural decisions, the question of how a particular architecture of a product can be represented in sufficient levels of detail will now be considered. An answer is required since this research aims to analyze a design process based on an understanding of the underlying product architecture. This section will therefore explore whether the available literature provides a means for clearly representing a particular product architecture. In particular, the work of Pimmler and Eppinger (1994) who proposed a taxonomy of interactions that is commonly used to model the architecture of a product will be explored. This approach will first be described and then discussed.

Architecture as interactions between building blocks

The starting point is the assumption that the above definitions of product architecture and modularity are very similar to Simon's hierarchical concept as described in section 2.1. Modular products can be compared with the concept of nearly decomposable systems where the interactions within the blocks are much stronger than the interactions between the blocks. The more integral an architecture, the stronger the interactions between the blocks become.

Based on this, Pimmler and Eppinger proposed a method for constructing a product architecture by illustrating the interactions within a product. They documented all physical elements (N.B. these are decomposed building blocks) of a product and identified all their interactions. They subsequently modeled all elements and interactions in a matrix – the elements were plotted identically in the rows and columns of the matrix and the interactions between them in the elements of the matrix. The search for relatively autonomous groups of physical elements then began. Their aim was to identify clusters of physical elements where the interactions within the clusters were much stronger than the interaction between the clusters. Once satisfactory clusters of physical elements had been found, each cluster formed one of the building blocks of the product. The matrix thus shows the building blocks of a product and their interactions.

Especially interesting is the way Pimmler and Eppinger defined interactions between building blocks. They recognized that building blocks may have different types of interactions since there are many different technical reasons behind a particular product architecture (as Ulrich demonstrates). Accordingly, they proposed a taxonomy of four types of interactions.

The following four types were identified:

- **Energy**: An energy-type interaction identifies the need for adjacency or orientation between two building blocks (or physical elements).
- **Material**: A material-type interaction identifies the need for materials exchange between two building blocks (or physical elements).
- **Information**: An information-type interaction identifies the need for information or signal exchange between two building blocks (or physical elements).
- **Spatial**: A spatial-type interaction identifies the need for adjacency or orientation between two building blocks (or elements).

Each interaction type can be rated as required, desired, indifferent, undesired, or detrimental for functioning. It should be noted that the first three types refer to physical exchange of energy, material, or information, respectively. For instance, the driving unit of an electric shaver requires energy (electricity) from the power supply to perform its function of providing...
rotation. To physically realize this, an electric wire must link both blocks. An example of a
detrimental exchange of energy is a driving unit that hampers the functioning of the power
supply because it vibrates too much. Other reasons include too much heat or radiation.
The spatial interaction type concludes the taxonomy with an indication of whether blocks
should be located in proximity to each other, as far from each other as possible, or somewhere
in between.

The work of Pimmler and Eppinger can be viewed in two ways. First as a promising way of con-
structing new architectures, and second as a clear way of visualizing a particular (existing)
architecture. We are mainly interested in the second aspect, which will be discussed below.

Discussion
In order to come up with a clear way of representing product architecture, the matrix
representation and the taxonomy of interactions will be briefly discussed.

– Matrix representation
The matrix representation offers a clear way of modeling interactions between building
blocks. Every interaction between each possible pair of building blocks can be captured and,
moreover, each possible type of interaction can be modeled. The traditional way of illustrat-
ing architecture (see Figure 2.12) is much more limited. In such a scheme of blocks and
functions it is very difficult to show the interactions between all possible pairs of blocks, and
not all interactions (i.e. the interfaces) are visualized. The modeling of interactions and
building blocks thus provides the perfect basis for depicting and analyzing a product archi-
tecture.

– The taxonomy of interactions
The proposed taxonomy of interactions has often been applied in recent research and serves
as a basis for architectural studies (Terwiesch & Loch 1999, Stake 1999, Blackenfelt 2000,
Sosa et al. 2000). In order to obtain an initial idea of whether it is suitable for this research,
the taxonomy will be compared with alternatives in the literature. The taxonomy will then be
compared with the work done by Ulrich.

Table 2.1 illustrates several alternative taxonomies that have appeared in the literature. It
is not the goal of this research to elaborate these in great detail, but it should be observed that
the taxonomies have much in common. They all draw a basic distinction between exchanges
between the three flows (energy, material, or information) and geometrical aspects, including
undesired or unintended effects or side effects. The aspects addressed within the taxonomy of
Pimmler and Eppinger are thus almost identical to the issues that have been distinguished by
other researchers in the field. However, compared to the others, it seems that Pimmler and
Eppinger pay slightly less attention to the geometrical aspects of their interaction types.
Table 2.1 Different types of interaction (based on Stake 1999)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Material</td>
<td>Material</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Energy</td>
<td>Energy</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Information</td>
<td>Information</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Spatial (=Proximity)</td>
<td>Incidental:</td>
<td>Geometry</td>
<td>Spatial (location + volume)</td>
</tr>
<tr>
<td></td>
<td>Geometric arrangement+</td>
<td></td>
<td>Attachment</td>
</tr>
<tr>
<td>Detrimental:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(side effects)</td>
<td>Side effects</td>
<td></td>
<td>Environmental (Side effects)</td>
</tr>
</tbody>
</table>

In the next section, the taxonomy of interactions will be discussed further and the taxonomy compared with the definition of product architecture.

2.3.3 Comparison

A comparison of the previous two sections (2.3.1 and 2.3.2) leads to a remarkable and important observation. The decisions that according to Ulrich determine a product architecture, and the two important features, mapping and interface coupling, are in fact difficult to capture within Pimmler and Eppinger’s four types of interaction. The most compelling question is how can taxonomy include a function that is mapped across two building blocks? A short glance at the taxonomy will probably be sufficient to show that this is not possible.

This concept is an important one in this thesis and will be further elaborated in Chapters 4, and 5. There, the benefits of being able to link the architectural three decisions with interaction types will become apparent, and a new taxonomy of interactions will be developed. This topic will not be pursued in the present section since the organizational principles also need to be taken into account when discussing the taxonomy.

At this point it will suffice to state that despite the whole engineering world being convinced of the validity of Ulrich’s architectural decisions, these are not clearly present within taxonomies that have been proposed to represent product architecture.

To sum up, the following points can be made:

- A matrix representation that illustrates building blocks and their interactions seems a very useful way of clearly representing an architecture.
- Since product architecture consists of several technical decisions there will be various types of interaction between building blocks.
- The available taxonomies in the literature are quite similar and cover a broad range of technical aspects, but have a remarkably poor correspondence with the architectural decisions that determine a particular product architecture.
2.4 The implications of architecture

The previous sections have gradually led to a definition of product architecture. Now is the time to finally demonstrate why product architecture is such a crucial issue. A brief summary of available studies that have extensively reported on the high impact of architecture on manufacturing firms will now be given. Furthermore, the reasons why some products are modular and others integral will be listed.

Product architecture has been a frequently studied topic in relation to many business topics. Many have produced very complete overviews of all implications and considerations. Ulrich 1995, Ulrich and Eppinger 1995, and Ulrich and Eppinger 2000, for instance, suggest that product architecture affects how variety is established within production, how change can be realized across subsequent generations of products, how building blocks can be standardized, the overall performance of products, and the management of product development. On the other hand, Erixon 1998, Stake 1999, and Blackenfelt 2001 argue that architecture impacts on product development, product variants, quality (testing), purchasing and after sales.

It is recognized by all authors that the design of modular products has a number of clear benefits.

- Modular products allow the production of a great variety of end products from a limited number of building blocks (Ulrich 1995, Erens 1996, Martin and Ishii 2000).
- Modular products facilitate changes to products once introduced (Baldwin and Clark 1997, Sanchez 1999c).
- Modularity allows for outsourcing of building blocks (Novak & Eppinger 1998).
- With today's pressure and increasing complexity there is clearly a trend in favor of more modular products (Baldwin & Clark 1997).

On the other hand, modularity also has a number of fundamental limitations or drawbacks:

- Too much modularity can make products look too much alike (Cutherell 2000).
- Modularity increases the risk of competitors copying the design (Cutherell 2000).
- Modularity is generally at the expense of unit cost and increases the volume (size) or weight of the product (Ulrich 1995, Whitney 1996).
- Modularity may be limited by the technology available (Whitney 1996).
- Designing modular products may be very difficult, be initially time-consuming (Ulrich, Sartorius, Pearson, Jakiela 1993), and depend on the capabilities of available designers within the company (Meyer & Utterback 1993).

Studies done by Henderson and Clark (Henderson & Clark 1990) (which will be returned to later in this thesis) prove that an established architecture is strongly embedded within the organization and the company’s way of working. In addition, Sanchez (1999a) argues that the entire ‘knowledge’ of a firm is strongly shaped by the architecture of a product.
Based on this brief summary of factors that have to be addressed within the context of product architecture, it is reasonable to state that architecture strongly affects and is affected by a firm's strategic considerations, and furthermore heavily influences how the company actually works.

Later in this research this knowledge will prove invaluable in understanding the causes of particular architectural decisions, but also in understanding that changing an architecture is not just a matter of techniques, but also affects the firm's entire policy.

2.5 Summary

This chapter's objective was to define the concept of product architecture and its underlying decisions, and to explore how a particular product architecture can be represented. Product architecture was defined according to the work done by Ulrich. Product architecture is:

· The arrangement of functions.
· The mapping from functions to physical building blocks.
· The specification of the interfaces among interacting physical building blocks.

The most important characteristic of product architecture is its amount of modularity:

· A modular product has one-to-one mapping between functions and building blocks, and its interfaces are de-coupled.
· An integral product has complex (N-to-M) mappings between the functions and the building blocks, and its interfaces are coupled.

In general, products are neither entirely modular nor entirely integral but rather somewhere in between the two extremes. They are called hybrid architectures. As a rule, the more the mapping is one-to-one and the more the interfaces are de-coupled, the more modular a particular product architecture is.

Furthermore, the importance of definitions of functions and building blocks in order to obtain a correct understanding and interpretation of the definition of architecture was stressed. The design methodology of Pahl and Beitz (1996) and the Axiomatic Design approach of Suh (1990) were discussed and their definitions compared. The following set of definitions was produced and will be used during the remainder of this research.

· A function is a design goal, describing what needs to be achieved (in line with AD). Where possible, functions should be conceptualized as transformations of energy, material, or information (in line with VDI). The VDI function concept is in fact a sub-set of the AD definition of functions.
· Building blocks physically fulfill functions. They locate a collection of detailed physical design parameters that specify the physical properties required to establish the functions.
· In order to find a solution for a function, generate-test cycles are required.
· A function cannot be decomposed without addressing its physical solution (at the same level of abstraction).
· Constraints are specifications that limit the range of possible solutions. Global constraints depend on many decisions and cannot be allocated to specific building blocks or functions.
These definitions and the engineering design models described in this chapter will provide the basis for the interpretation of product architecture in an organizational context.

It was then argued that a particular architecture can be clearly presented as a set of interacting building blocks depicted in a matrix. Various technical reasons why building blocks may interact are captured within a taxonomy of interactions that distinguishes several types of interactions. The best known is the taxonomy by Pimmler and Eppinger who propose 4 types of interactions. It was, however, concluded that this taxonomy has a poor correspondence with the original definition of product architecture by Ulrich. This aspect will be revisited in Chapters 4 and 5.

Finally, the underlying contingencies of product architecture were described. This knowledge is valuable for understanding the causes of particular architectural decisions, but also for understanding that changing an architecture is not just a matter of techniques but also affects a firm’s entire policy.