TU/e EINDHOVEN UNIVERSITY OF TECHNOLOGY

# Architectures for product families

*Citation for published version (APA):*
Erens, F. J., & Verhulst, K. (1997). Architectures for product families. *Computers in Industry*, *33*(2-3), 165-178. https://doi.org/10.1016/S0166-3615(97)00022-5

*DOI:*
10.1016/S0166-3615(97)00022-5

*Document status and date:*
Published: 01/01/1997

*Document Version:*
Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

*Please check the document version of this publication:*

• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

*General rights*
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the "Taverne" license above, please follow below link for the End User Agreement:

www.tue.nl/taverne

*Take down policy*
If you believe that this document breaches copyright please contact us at:

openaccess@tue.nl

providing details and we will investigate your claim.

Download date: 04. Oct. 2023

# Architectures for product families

**Freek Erens[1] & Karel Verhulst**

E-mail: freek@erens.net

## Abstract

This paper analyses the role of architectures in the development of product families. It is argued that modularity and integration are conflicting requirements that are best solved using product architectures. To describe a product family, three domains can be recognised: the functional domain, the technology domain and the physical domain. Each domain is characterised by both a hierarchical view (the product structure) and a non-hierarchical view (the product architecture). Especially, product architectures are important for developing product families. Using the domains, a definition for a product family is given. Deviations from the product family can be described with a special modelling language, named the generic product structuring concept.

## 1. Introduction

The advent of the buyers' market has imposed a necessity on manufacturing companies to suit individual customer requirements. Companies have answered this need by offering a large variety from which customers can choose their ideal products. Preferably, these products have many common modules to reduce the efforts in development and increase the economy of scale in manufacturing. Still, specific modules must be developed to create the necessary differentiation of products for the individual customers and market segments. The proportion of common and specific modules must be chosen such that profits are maximised. Modularity corresponds to the cost / variety trade-off.

A second consequence of the buyers' market has been an increasing pressure on manufacturing companies to increase product performance while simultaneously reducing costs. Prominent examples of this phenomenon can be found in consumer electronics and more recently in personal computers. Technological progress in chip development and software engineering has made it possible to integrate an increasing number of user functions in a decreasing number of modules. However, the pursuit of integration can conflict with modularisation, being necessary to meet the challenge of product variety. Integration corresponds to the cost / performance trade-off.

Modularity and integration can be conflicting requirements. It is argued that complexity resulting from these conflicting requirements can be answered with a product architecture (see Figure 1). This paper focuses on architectures for product families as especially product families cope with the issue of achieving modularity and integration simultaneously.

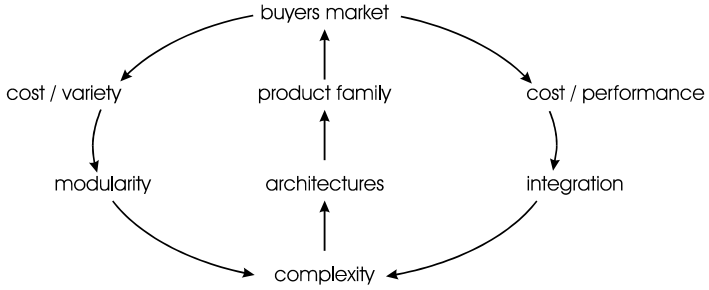


*Figure 1. Architectures for product families*

The structure of this paper is as follows. Section 2 discusses the role of product models in different development domains, after which section 3 focuses on the importance of product architectures. Then, section 4 explains how product models are connected in the development process. The issue of modularity and integration is discussed in section 5, after which section 6 defines the notion of a product family in relationship to development domains, product architectures, modularity and integration. Finally, section 7 discusses deviations from the product family concept and proposes a modelling language that captures a large variety of product variants in a non-redundant way.

## 2. Domains and product models

In the development process, many different product descriptions can be recognised, although most product descriptions can be classified in three categories [Albano, 1992][Andreasen, 1987][Erens and Verhulst, 1995]. These categories are represented by *product models* which act as a backbone for the combined product information. The product models that are shown in Figure 2 are used by different business functions and in several phases of development to organise product information in a structured way, including hierarchical and non-hierarchical dependencies:

- The *Functional Model* is a consistent description of the functionality of a product. It is strongly related to the purpose of the product. Usually, the Requirements Specification, created by product management, is an important input for this model.
- The *Technology Model* is a consistent description of the application of technologies (that is solution principles) to ensure the operation, but not necessarily the manufacturing, of the product. Development creates most of the information structured in this model.
- The *Physical Model* is a consistent description of the physical realisation of a system. It is strongly related to the construction of the product. Manufacturing sets conditions for this realisation in order to guarantee an easy assembly operation without compromising the quality level or cost level.

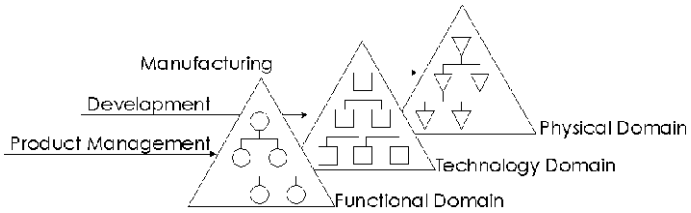


*Figure 2. Product Models*

Figure 2 depicts the contribution of product management, development and manufacturing to respectively the Functional Model, Technology Model and Physical Model. The depicted product models are independent of the hierarchical level of the product: systems, sub-systems as well as components can be represented by these models. The relationships between these models are discussed in section 4.

The following sections elaborate on these three domains using an example of a medical system. Medical equipment is complex, not only because of the use of advanced technologies, but also because of the combination of diverse technologies in one system. Technologies to control the position of the patient, the X-ray source and often several X-ray detectors on precise locations are to be combined with technologies that control the optimal functioning of an X-ray tube, an image intensifier, a film transport mechanism or video chains. Figure 3 gives an example of a physical view on a cardio-vascular system. This system is used for examining a patient's heart and vessels. The complexity of development originates from the fact that almost each physical assembly uses a variety of technologies to provide a user function.

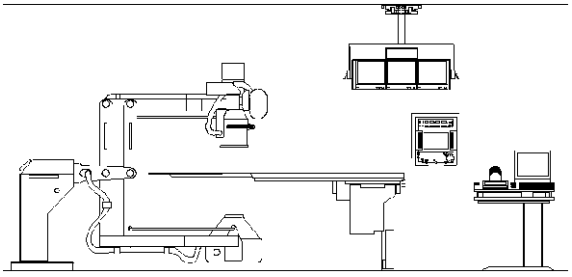

*Figure 3. Cardio-vascular system*

### 2.1 The Functional Model

A consistent description of the set of functions of a system is given in the Functional Model. It comprises the hierarchical structure of functions and the interfaces between these functions. The functional requirements are primarily listed in the Requirements Specification (RS) in a textual form. In a second stage they are converted into a more formal description. This is to establish dependencies in an explicit form. A structured analysis checks for completeness and consistency in the functional model [Hatley and Pirbhai, 1987]. This analysis of functional requirements has to separate functionality from technology. In the analysis, functionality is described in a hierarchical decomposition of functions to cope with complexity (see Figure 4).

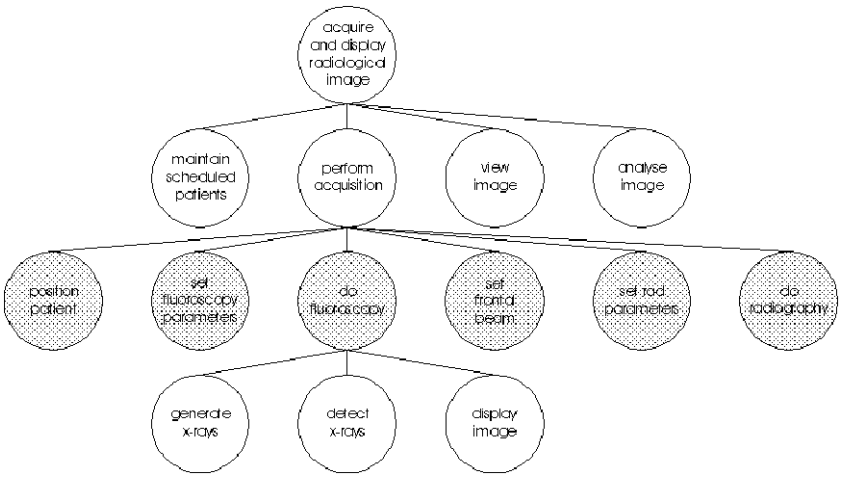


Figure 4. Functional structure

The Functional Model represents the intention of the requirements and guarantees an unambiguous realisation in the design process. Each level describes the functions and their dependencies and interactions that are relevant on that level. Design decisions are taken on one level of abstraction at a time. Figure 5 gives an example of one such a level, *perform acquisition*, in the hierarchical functional structure of a medical system. Functions are depicted as bubbles. Data dependencies between functions are depicted with arrows.

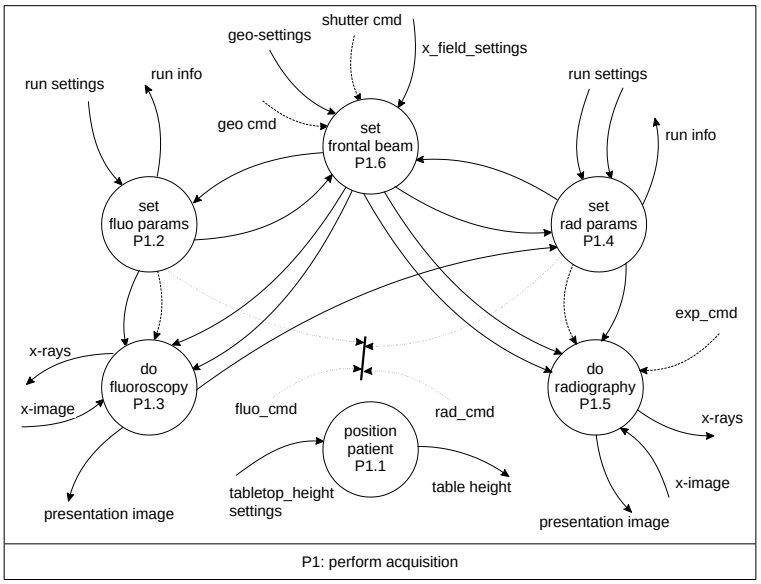


Figure 5. Functional architecture

## 2.2 The Technology Model

The Technology Model is a consistent description of all technologies that are applied in a system. It comprises the decomposition of technology modules and the interfaces between technology modules. A block diagram describes the technological structure of the system and assists the organisation of development projects. A simplified example of the technology decomposition is given in Figure 6.

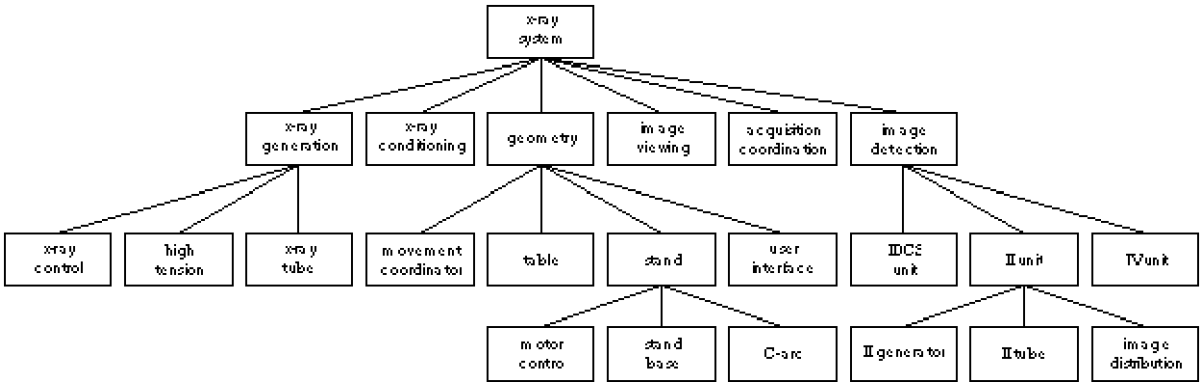


Figure 6. Technology structure

User functions are realised in a combination of modules. Similar to the development of the Functional Model, decisions with respect to the Technology Model are taken on one abstraction level at a time. The technology architecture describes the operation of a system by its modules and interfaces between those modules. In Figure 7, a technology architecture is given for the "X-ray system" on level one of the hierarchical technology structure of Figure 6.

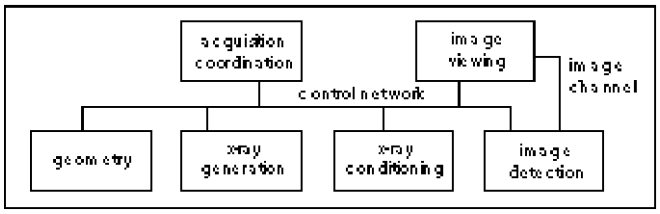


Figure 7. Technology architecture

The technology architecture consists of several modules and their interfaces. Modules are physical or logical units in which one or more functions of the Functional Model are realised, thereby taking into account constraints that have been formulated in the Requirements Specification (for example constraints on the reuse of technologies). Examples of modules are: a processor, a display, a video-board, a motor, etc. An interface is a specification of the interaction between modules. This can be a computer bus, an optical path or a mechanical connection.

### 2.3 The Physical Model

The consistent description of a system's parts and assemblies is named the Physical Model. The modules of the Technology Model are physically implemented in assemblies that can be manufactured and serviced. Design decisions in the Physical Model are taken on one abstraction level at a time. The physical architecture is described in assembly drawings. Figure 8 gives an example of the hierarchical physical structure, while Figure 9 positions the parts of the stand in the physical architecture.

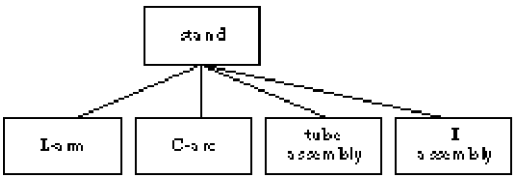


Figure 8. Physical structure

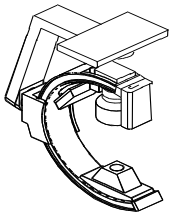

Figure 9. Physical architecture

The fact that medical equipment is manufactured with some volume is the reason for a difference between an ideal Physical Model for manufacturing and the Technology Model that is created in development. Eventually, the Physical Model is a compromise between the requirements of development, manufacturing, service and other parties that have an interest in the physical implementation of the product.

## 3. Product architectures

The previous section discussed product models describing a product from a functional, technological and physical perspective. To reduce the complexity of the description, and thereby the possibility to understand and solve the design problem, products are decomposed into components. Decomposition is a recursive activity, which eventually results in a set of primitive design objects, either functions, technology modules or assemblies.

The composition of a product from a number of component products is a product architecture. It describes the components, together with their interfaces and operation. Each level in the product hierarchy has its architecture. Depending on the type of components, we speak about a functional, technology or physical architecture.

Although a product architecture can be the result of a decomposition activity, it can also be the starting point of development. Increasingly companies define product architectures before the development of products is commenced. There are several reasons for this:

- *Stability*. Interfaces between components are set in order to reduce the effect of changes in one component for related components. In such a way, a product architecture creates a stable environment in which components can be developed in parallel with managed risks. However, the value of a product architecture is related to the maturity of the product. Innovative products, for example, are characterised by frequently changing requirements and solution principles. This reduces the possibility to use a detailed product architecture, although the value of a more generic product architecture, leaving much flexibility for the remainder of the development process, can still be significant;
- *Communication*. Stability also reduces the need for communication between developers, especially if they are responsible for different components of the product. Communication does not only concern verbal discussion, but also written documentation, particularly if design knowledge is needed later in time. A product architecture should be a stable framework for associating development documentation;
- *Learning organisation*. Stability is a prerequisite for learning. Therefore, companies are often organised around a product architecture. Both the company's organisation and the product architecture may be reinforced by the process through which design problems are solved. This improves the possibility to solve similar problems, but at the same time inhibits the development of new products that depart significantly from known products. A breakthrough product therefore not only requires a new product architecture, but also a change of the development organisation;
- *Commonality and variety*. A predefined product architecture creates the possibility to replace components with components that have identical interfaces. Combinations of these components cater for the diverse requirements of customers. Preferably, all components can be combined in arbitrary ways to improve the ratio of commercial and technical variety. Some components, however, will not have variants and are therefore common for the product family. These components are a relatively stable factor in the design;
- *Reuse and upgrading*. A product architecture is usually more stable than its components. This can be used to create a new version of the product by replacing some of the components with an upgraded version. A number of existing components is reused to reduce the time, effort and risk of developing a completely new product. However, incremental development requires a good insight in the life-cycles of the different components and the product architecture in which these components must fit;
- *Competitive control.* Companies that control architectural standards have an advantage over other vendors. Since they control the interfaces and often the critical components, they are better positioned to develop products that maximise the possibilities of the architecture. Furthermore, they can discipline competing vendors that offer components that are compatible with the architecture. Architectural controllers try to increase the competition among these competing vendors, which results in a price reduction of the overall product and an increased market share for the architectural controller.

The above issues share a common theme, which is separating the stable and changeable aspects of development. The stable aspects of development are used to create a framework within which the

changeable aspects can be executed in relative isolation, for example by component developers and external suppliers. This paper pays most attention to those changeable aspects that create product variety, although it is our contention that the principles developed have a general applicability.

To understand the issue of product variety, it is important to see how product architectures in different domains are related. Therefore, the next section gives a brief introduction to a design method that discusses the evolution of different representations of the artefact. This not only means detailing product descriptions in the different domains, but also creating relationships between these product descriptions.

## 4. Coupled domains

The Requirements Specification of a development project is usually given in the functional domain, after which solutions are sought in the technology domain and the final implementation is realised in the physical domain. The different domains must co-operate in the development process and consequently the relationships between the product models in these domains must be maintained. An important relationship for understanding the issue of product variety is the *allocation relationship*. Functions are allocated to technology modules and these in turn are allocated to physical assemblies. This section discusses a design method in which the allocation relationship plays an important role.

Important in this design method is that once functions are defined at a given level of the functional hierarchy, they cannot be decomposed independently of the evolving hierarchies in the technology domain and the physical domain. Consequently, an iterative scheme of decomposition and allocation between the functional domain and technology domain must be used to incorporate new information regarding functions and solution principles. This process suggests a zigzag pattern. A zigzag pattern also governs decomposition and allocation between the technology domain and the physical domain.

Decomposition and allocation are two important steps in the design cycle that is depicted in Figure 10. This design cycle is very similar to the *productive reasoning model* of March [1984] and Cross [1989]. In total, there are four design steps that can be applied on all levels of the domains' product hierarchies:

- *Decomposition* is adding detail to a product model. On the highest level, the three models describe the same product. However, on that level, product descriptions are still very generic and cannot be discriminated. Furthermore, when it is difficult to map functions onto technology modules, these functions must be detailed until they can be allocated;
- *Allocation* is the creation of relationships between elements of different product models. For example, several functions are allocated to a technology module. In the same way, several modules are physically realised in an assembly of the Physical Model;
- *Composition* is combining elements of a product model, for example modules of the Technology Model or assemblies of the Physical Model;
- *Validation* is a check on the realised quality level of the product model, by relating it to a previous product model. For example, a composed module is validated against the original function, while an assembly is validated against the original technology module.

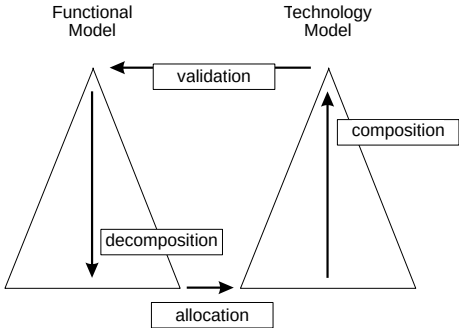


*Figure 10. Four elementary design steps*

The allocation and validation steps produce relationships between product architectures and consequently result in coupled domains. The decomposition and composition steps are applied to make these allocation and validation steps possible. The result is a set of harmonised product descriptions that can be unambiguously understood in the development process.

The next section builds on the notion of coupled domains. It discusses how the allocation of functions to technology modules, and of technology modules to physical assemblies, determines the level of modularity or integration of a product.

## 5. Modularity and integration

Modularity and integration are important criteria in developing product families. Modularity corresponds to flexibility and changeability, while integration corresponds to stability and optimisation. These contradictory requirements are largely determined in the early phases of development when the architecture of a product family is defined.

Both modularity and integration can be defined in terms of the allocation process. In general, a modular design is considered to be a design in which a restricted number of functions is allocated to a module, or a restricted number of modules is allocated to a physical assembly. The opposite of a modular design is an integrated design. Focusing on the allocation process from the functional domain to a particular level in the technology domain, there are four different possibilities:

*1:1* One function is allocated to one technology module. This is a modular design;

*1:N* One function is mapped to several technology modules. This distribution of a function over several modules results in an integrated design on that level;

*N:1* Several functions are allocated to one technology module. Again, function sharing increases the level of integration;

*N:M* Several functions are allocated to several technology modules. Functions are distributed and shared, thereby further increasing the level of integration.

Both 1:N and N:M mappings are ambiguous as it is not clear which part of a function is realised in a technology module. Therefore, both mappings must always be based on 1:1 or N:1 mappings on lower levels of the product hierarchy;

As stated before, modularity corresponds to flexibility and changeability. It is an effective mechanism to upgrade and reuse existing functions, modules and assemblies. Reuse multiplies the effectiveness of human problem solving by ensuring that the extensive work or special knowledge used to solve specific development problems will be transferred to as many similar problems as possible. This reduces *initial costs*, especially in product development.

In contrast, integration corresponds to stability and optimisation. In an optimised design, a large variety of functions is realised with a limited number of components. The initial costs of such a multi-functional design are usually higher than the costs of a modular design, but the *operational costs* (in manufacturing and service) are relatively low due to this limited number of optimised components. However, integration requires a stable environment and can therefore best be applied in mature products. The trade-off between modularity and integration is shown in Figure 11.

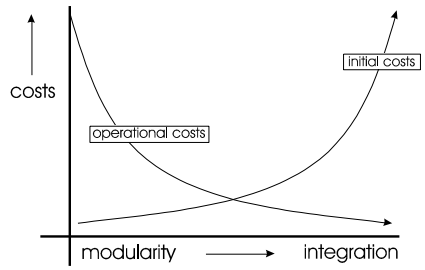


*Figure 11. Initial costs versus operational costs*

The decomposition process, and consequently the level of modularity or integration that can be achieved in a design, is very dependent on the product architectures in the different domains. Product architectures govern the process of allocating functions to modules and assemblies. Therefore, important considerations with respect to modularity and integration should be made in the early phases of the design process, where the product architectures in the different domains are determined.

Ulrich ea. [1990, 1991] state that most architectures evolve from being modular to being integrated. In later stages of the product life-cycle, when the interactions between different aspects are better understood, the architectures in the different domains should enable *function sharing* in order to make the design more efficient. According to Ulrich ea., the mechanism for function sharing in mechanical development is based on the fact that:

> *" it is easier to think about new problems if they are decomposed in a modular fashion, and partly because in the initial stages of product development, engineers want to be able to work independently on different aspects of the design.* (…) *Of the infinite properties of a structural element, only a small set is relevant to the behaviour the designer intends for that element. In addition to the primary properties of a structural element that provide that element's intended function, there are many secondary properties that are incidental to those that implement the intended function. The key idea that allows function sharing to be achieved by a design procedure is that these secondary properties of structural elements can be exploited. By recognising and exploiting secondary properties of one element, neighbouring elements can be eliminated from the design ".*

Modularity and integration seem to be conflicting requirements. However, as Figure 12 shows, both can be achieved simultaneously by reducing a third factor, namely the *design margin*. The design margin is the freedom that a designer maintains in the development process to allocate less critical requirements in a later stage. The remaining flexibility of the design is then used to find any solution that meets these requirements. Increasing both modularity and integration reduces flexibility and the possibility to postpone the allocation of less critical requirements. The early consideration of modularity and integration is reflected in the product architectures, which therefore reduce the design freedom in the remainder of the development process. However, reducing the design margin requires a certain maturity of the product family as more requirements have to be considered simultaneously.

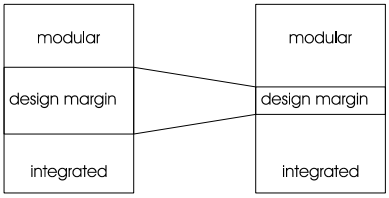


*Figure 12. Design margin*

Product families need architectures in which both modularity and integration are considered. Modularity is necessary to offer a large product variety, while integration is needed to improve the cost performance ratio. The complexity of this balancing issue is discussed in the next section.

## 6. Product families

Today's buyers market is a break with the past. The sellers market of the fifties and sixties was characterised by high demand and a relative shortage of supply. Firms had to choose between being innovative speciality businesses or being efficient mass producers manufacturing large volumes of identical products. According to Pine, Victor and Boynton [1993], this old competitive dictum was grounded in the well-substantiated notion that the two strategies required very different ways of managing, and, therefore, two distinct organisational forms. Nowadays, the buyers market is forcing companies, making high-volume products, to manufacture a growing range of products tailored to individual customer's needs but at the cost of standard mass-produced goods.

Mass-customisation is a major problem for manufacturing companies. A trend that enforces this problem is decreasing life-cycles of products. Together, these cause an increase of *product density*, [Erens and Breuls, 1995] that is, the number of product variants that are introduced over the life-cycle of the product family (see Figure 13).

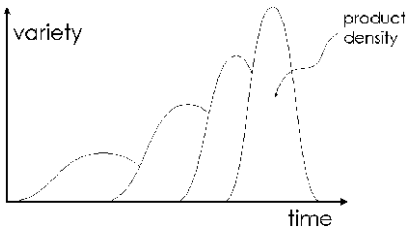


*Figure 13. Increasing product density*

This increase is even enforced by the improved economic life-time of products installed in the field. Although the life-cycle of product types is decreasing, the economic and technical life of the installed products is not. Therefore, the variety of products that are actually alive in the market, is increasing in proportion to the new product launches.

A high product density requires synergy in development, which can be accomplished by developing product families. Product families are a means to improve the commercial variety while limiting the development, manufacturing and servicing efforts. Product families are based on the technique of reuse. However, a prerequisite for reuse is a strong link between the company's strategy and the individual product development projects. Modules can only be reused if customer wishes are anticipated in an early phase so that a product family architecture with stable interfaces, in which these modules are exchanged, can be created.

**Definition of a product family**

This paper defines a product family as a product with identical internal interfaces, i.e. interfaces between the product's components, for all variants in each domain. Interfaces must be standardised in each of the functional, technology and physical domain to allow the full exchange of components.

The above definition is explained with an example. Figure 14 mentions three products (i.e. technology modules) in the technology domain: product 1, product 2 and product 3. Each product has two components: C0 - C1, C0 - C2, and C3 - C4. Product 1 and product 2 are variants of the same product family if they do not only have identical internal interfaces in the technology domain but also in the functional domain and the physical domain:

- *Technology domain*. The interfaces between the components C0 - C1 and C0 - C2 are identical for both products (see Interface 1 in Figure 14). Product 3 does not belong to this product family as it has a different internal interface (Interface 2) between the components C3 - C4. In other words, component C3 can only be used in combination with component C4 and not in combination with the components C1 or C2. This limitation to make arbitrary combinations of components reduces the number of different parent products. The possibility that the components C1, C2 and C4 are a product family again does not change this.

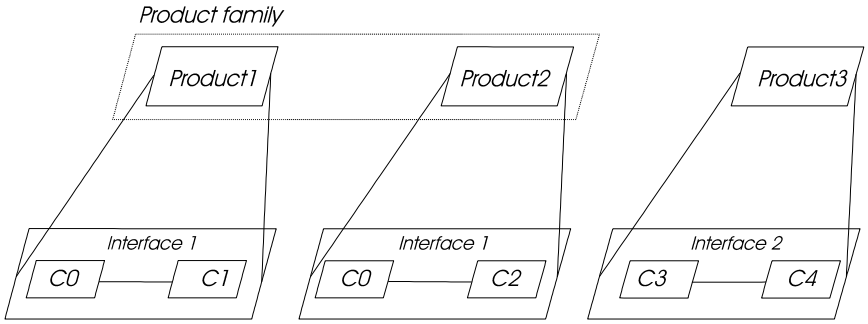


*Figure 14. Example of product variants*

- *Functional and physical domain*. The interfaces between the corresponding functions in the functional domain and physical assemblies in the physical domain must be identical for Product 1 and Product 2. A similar figure as is shown above can be created for both the functional domain and the physical domain.

From the definition of a product family follows that not every product in the decomposition hierarchy of a product family is a product family again. Whether a product is a product family or not solely depends on the interfaces of its components. It does not depend on the individual architectures of the product family's components, neither does it depend on the external interfaces of the product family itself. In terms of the example, it is not important whether the components C1 and C2 belong to the same component family as long as they have identical external interfaces with component C0.

**Co-ordinated product architectures**

The standardisation of interfaces in one domain makes it possible to create arbitrary combinations of components in that domain. However, the development of a product family requires co-ordination of product architectures in different domains. The standardisation of interfaces, and consequently the reuse of components, asks for anticipating potential customer wishes in an early phase so that the product architectures in the functional, technology and physical domain can take these into account.

The issue of co-ordinated product architectures is related to the allocation process between domains. A physical assembly can only be easily isolated in its context if one or a limited number of technology modules is allocated to this assembly. This in turn requires the allocation of one or only a few functions to each variable technology module. In contrast, the stable functions, technology modules and physical modules can be highly integrated to improve the cost performance ratio.

A product family is characterised by 1:1 and N:1 allocation relationships between domains. Both 1:N and N:M allocation relationships disqualify a product as a product family: the distribution of a function over several technology modules implies that a technology module can only be chosen together with several other technology modules. In other words, there are dedicated interfaces between technology modules and therefore not all combinations of technology modules are allowed. This conflicts with the definition of a product family. In contrast, a 1:1 or N:1 allocation of functions to technology modules preserves the possibility to make arbitrary combinations of technology modules. If N functions are allocated to one technology module, then the interfaces between these functions are also allocated to this one module. Therefore, they do not harm the standardised interface this module has with other technology modules.

Figure 15 gives an example of co-ordinated product architectures. The grey circle, square and triangle denote respectively a function, a technology module and a physical assembly. They are connected with 1:1 allocation relationships. Together, the function, technology module and physical assembly create a modular aspect of the design.

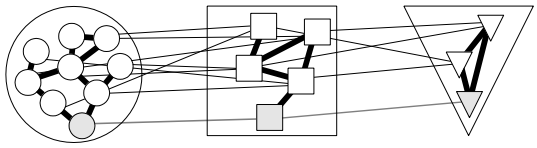

*Figure 15. Example of co-ordinated product architectures*

Standardisation of a product family's interfaces in all three domains requires synchronisation of these architectures in the three domains. This is realised as described above. The components that result from this have an internal architecture that is independent of the architecture of the product family. The development of the product family and its components is decoupled: components are developed within the context of the product family's architecture. These "levels" in the hierarchical product structure are an effective mechanism to control design complexity. A deliberate use of this contributes to a more stable development process.

**The origin of variety**

The variety of a product family depends on the variety of its components. In the example of Figure 14, the components C1 and C2 have a common interface with C0, which makes Product 1 and Product 2 variants of the same product family. This does not imply that also the components C1 and C2 belong to the same product family. There are two possibilities:

- The components C1 and C2 are variants of the same component family. In that case, they have standardised internal, as well as external, interfaces in all domains;
- The components C1 and C2 are not variants of the same component family, but share their external interfaces. The internal interfaces are different in one or several domains.

If C1 and C2 are component families again, their variants do not individually determine the external interfaces of C1 and C2. Else, Product 1 and Product 2 would not belong to the same product family. If the external interfaces of, for example, C1 cover all variety of C1, the design effort is seriously reduced. It is not needed to design dedicated interfaces for variants of C1, neither is it necessary to

design component variants at the “other side” of the interface. This is an important advantage of developing product families.

If all products in a decomposition hierarchy are product families, the variety of the end-product results from the variety of the primitive families. All variants of the end-product have an identical product structure, which increases efficiency in development and manufacturing. However, as the differences between the variants of the end-product are based on relatively small differences between the variants of the primitive families, the differences between these end-product variants are relatively small as well. This possibly reduces the scope of the product family towards the market.

In an ideal situation, the scope of the product family is large in the functional domain and relatively small in the technology and physical domain. The functional domain corresponds to the viewpoint of the customers and should have a large variety of functions to discriminate between the different requirements of these customers. The technology and physical domain define the realisation of these requirements and should preferably have a limited number of technology modules and assemblies to reduce both initial and operational costs. The development of a product family is characterised by the use of N:1 allocation relationships between domains. As a consequence, the scope of a product family is smaller in the technology and physical domain than in the functional domain.

## 7. Deviations from the product family

The product family of the definition is characterised by having identical internal interfaces for all variants in all domains. In real manufacturing situations, for example the manufacturing of medical equipment, most products do not strictly adhere to this definition. Nevertheless, many of these products are called product families as a large variety of end-products is created from a limited variety of components. However, there are components that cannot be combined to create new product variants, in other words, there are components that can only be applied in combination with a limited number of other components. This is the result of an allocation process in which a variable function is distributed over several components (or a variable technology module is materialised in several physical assemblies).

The existence of 1:N (or N:M) allocation relationships asks for a special product modelling language. A large variety of products sharing many components must be described in a non-redundant way. This modelling language is named the generic product structuring and meets the following requirements:

- First of all, the modelling language should enable the decomposition of a product into its components, each being either a product family again or a set of products with identical external interfaces;
- Secondly, the modelling language should enable the co-ordination of dependent component variants, i.e. components that can only be chosen in their combination. These are the actual deviations from the product family concept.

The generic product structuring concept distinguishes two different types of products: primitive generic products and compound generic products (see Figure 16). Both types of products do not necessarily comply with the definition of a product family as given earlier in this paper.

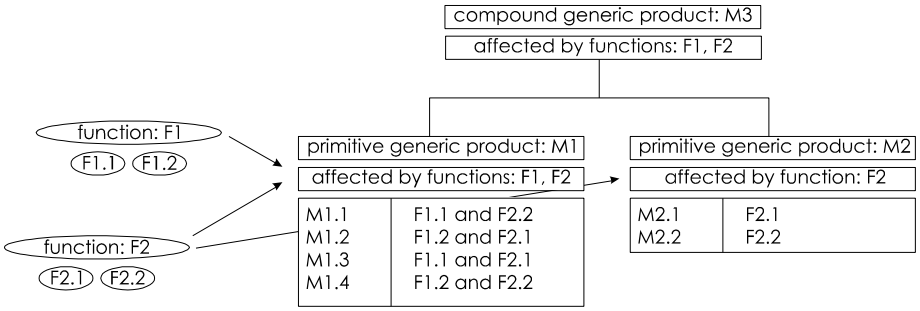


*Figure 16. Example of generic products*

The first type of generic product cannot be further decomposed into generic products. It has one or more variants. These primitive variants are the origin of variety. The second type of generic products is composed of primitive generic products or other compound generic products.

If a primitive generic product is a technology module, its variants are selected using the variants of the functions that are allocated to this module. Figure 16 shows that the variants of M1 are selected using the functions F1 and F2. Technology module M2, which has only two variants, is affected by F2.

A function can be distributed over different technology modules. The need for a function variant then results in the selection of several module variants. In Figure 16, function F2 is distributed over M1 and M2. This results in the fact that module variant M1.1 can only be applied in combination with module variant M2.2. This distribution of the function F2 requires a co-ordination mechanism. A distributed function is co-ordinated at the first common parent of the technology modules that are affected by this function, in the example module M3.

A elaborated description of the generic product structuring concept is given in Erens [1992], Hegge [1995] and Wortmann [1995]. The use of the concept in production control systems is described by Bottema [1992].

## 8. Conclusions

This paper described the use of product families to meet the challenge of offering a large commercial variety with limited development and manufacturing efforts. Although some attention is given to the design process, this paper mainly discusses the structural aspects of development, namely the definition of a product family in different domains, including the relationships between these domains. From this discussion, the following conclusions can be drawn:

- Product architectures are essential to separate the stable and variable parts of design. The stable aspects create a framework within which a variety of products can be developed;
- The standardisation of interfaces in one domain improves the possibility to combine components in such a way that a large variety in that domain is created;
- The standardisation of interfaces in three domains, and consequently the existence of N:1 allocation relationships, reduces the number of technology modules and physical assemblies that is needed to create commercial variants in the functional domain;
- The architecture of a product family is decoupled from the architectures of its components. The variety of these components has no consequences for the external interfaces of these components, which reduces design complexity;

- Deviations from the product family definition (i.e. distributed functions and distributed technology modules) are effectively controlled with the generic product structuring concept. This modelling language describes a large variety of products in a non-redundant way.

An unresolved research question is the maturity of a product family in relationship to the level of modularity and integration in the three different domains. Personal observations show that, over the life-cycle of a range of succeeding product families, integration starts in the physical domain, is then continued in the technology domain and eventually includes the functional domain. The authors plan to pursue further research in this subject.

## 9. Literature


Albano L.D., Suh N.P.
*Axiomatic Approach to Structural Design*, Research in Engineering Design, 1992, 4:171-183, Springer-Verlag New York Inc.

Andreasen M.M., Hein L.
*Integrated Product Development*, IFS Publications Ltd / Springer Verlag London, 1987

Bottema A., Tang L. van der
*A Product Configurator as Key Decision Support System,* in*:* Integration in Production Management Systems, Pels H.J. and Wortmann J.C. (editors), Elsevier Science Publishers, IFIP, 1992

Cross N.
*Engineering design methods*, John Wiley & Sons, 1989

Erens F.J., Hegge H.M.H., Veen van E.A., Wortmann J.C.
*Generative bills-of-material: an Overview*, Integration in Production Management Systems, Pels H.J. and Wortmann J.C. (editors), Elsevier Science Publishers, IFIP, 1992

Erens F.J., Verhulst K.
*Designing Mechatronic Product Families*, in: Proceedings of the WDK workshop on Product Structuring, Tichem M., Storm T., Andreasen M.M, and MacCallum K.J. (editors), Delft University of Technology, June 22-23, 1995

Erens F.J., Breuls P.
*Structuring Product Families in the Development Process*, Proceedings of ASI'95, Lisbon, Portugal, 1995

Hatley D.J., Pirbhai I.A.
*Strategies for Real-Time System Specification*, Dorset House Publishing, 1987

Hegge H.M.H.
*Intelligent Product Family Descriptions for Business Applications*, Thesis of Eindhoven University of Technology

March L.J.
*The logic of design*, In N. Cross (editors), *Developments in Design Methodology*, John Wiley & Sons, 1984

Pine B.J.
*Mass Customization*, The New Frontier in Business Competition, Harvard Business School Press, 1993

Ulrich K.T., Seering W.P.
*Function Sharing in Mechanical Design*, Design Studies, Vol. 11, No. 4, 1990

Ulrich K., Tung K.
*Fundamentals of Product Modularity*, DE-Vol. 39, Issues in Design Manufacture Integration, ASME 1991

Wortmann J.C., Erens F.J.
*Control of variety by generic product modelling*, Proceedings of the First World Congress on Intelligent Manufacturing Processes & Systems, Puerto Rico, February 1995