

CoFluent Design | White Paper

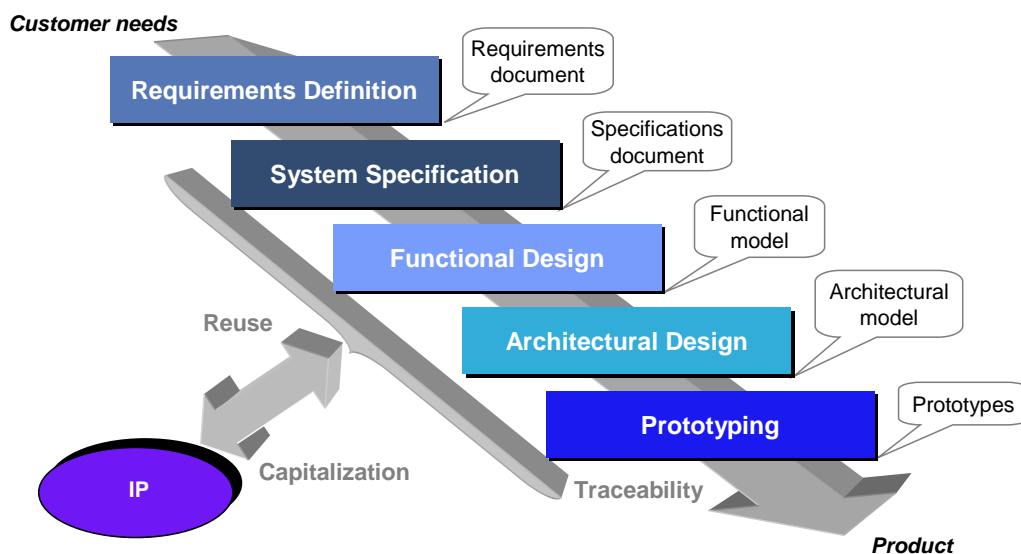
The MCSE Methodology Overview



The MCSE methodology (*Méthodologie de Conception des Systèmes Electroniques*, also known as CoMES – *Co-design Methodology for Electronic Systems*) is based on a top-down design process.

A design process consists of a series of steps that transform an input (*specifications*) into an output (*solution*) passed as input to the next step. The organization of the steps is done according to a *design process model*. Several process models have been suggested and experimented: waterfall model, V model, spiral model, contractual model, etc. If any design process model can be defined and applied, it is crucial that the meaning, goal, inputs and outputs for each step are clearly defined.

The MCSE design process organizes the required steps in a top-down manner. Designers can work simultaneously on several steps given that they respect dependencies between steps and design choices. The flow is not continuous as additional verification activities, resulting in a backward flow, are necessary to correct and/or enhance solutions.



The MCSE design process

System designers proceed according to a minimum of 5 steps:

1. **Requirements definition**
2. **System specification**
3. **Functional design**
4. **Architectural design**
5. **Prototyping**

Design step	Purpose	Inputs	Outputs
1. Requirements definition	Understanding all interested parties' needs and documenting these needs as written definitions and descriptions.	Customer needs	Requirements document
2. System specification	Representing a purely external view of the system. Completely modeling the behavior of the system within its environment and listing all non-functional constraints.	Requirements document	Specifications document
3. Functional design	Finding a suitable internal logical (or functional) and technology-independent representation of the system from an application-oriented viewpoint.	Specifications document	Functional model
4. Architectural design	Defining the detailed system's physical (or hardware) architecture and the organization of the software application on each programmable processor.	Functional model Performances/ costs constraints	Architectural model
5. Prototyping	Developing an operational system prototype in terms of hardware and software implementations.	Architectural model Technological & economical constraints	Prototype

In addition, the MCSE design process enables design traceability as well as IP capitalization and reuse at every design stage.

Forward and backward traceability between steps is enforced to:

- Capture the relations between initial requirements and all subsequent design models;
- Manage potential changes in requirements.

Capitalization and **reuse** are essential activities for a correct and efficient use of IP components. Reusing components is useful during functional and architectural design, but also during prototyping. It helps designers shorten the design process.

It is facilitated in two ways:

- Outside-in: identifying external functional or architectural components that satisfy the required functionality and are interconnectable;
- Inside-out: identifying internal components of the solution under design to be reused in other projects.

To be reused, a component needs to be well-defined, correctly encapsulated, validated and conform to an interchange standard.

Requirements traceability is potentially a one-to-many relation between a requirement and elements of the design. It implies the ability to follow the whole design process in forward and backward directions. A correct record of traceability between requirements and system components enables customers and project managers to monitor progress in the project.

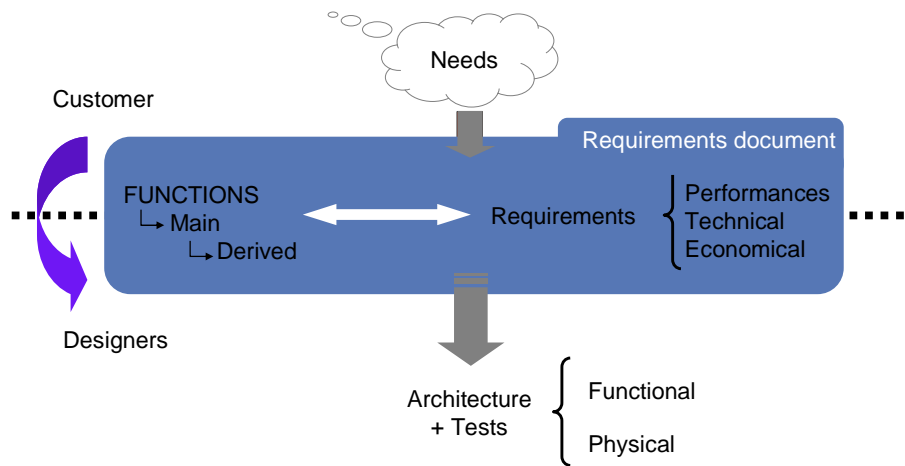
Requirements management deals with evolutive requirements. Modifications, changes, improvements and corrections in requirements are inevitable. Taking modifications into account is facilitated by a clear procedure applied to the whole design process and by appropriate tools.

1. Requirements Definition

The requirements definition step involves understanding all interested parties' needs and documenting these needs as written definitions and descriptions. The objective is defining *what problem the system has to solve*. First, designers analyze the environment in which the system operates, then the required system itself.

There are no formal languages currently available that can fully describe system functionality and requirements. Domain-expert natural language is used to allow exchanges between multiple stakeholders with various cultures and field-knowledges around complex problems.

The resulting requirements document clarifies and formalizes the relative responsibilities of the customer and designers and act as a contractual document between them. Designers are guided by the contents of this document during the whole design process since it defines the deliverables. As the requirements document is used as a baseline reference during the final system certification, it must be verified and validated by all stakeholders.



Requirements definition step

Defining customer requirements consists in finding, ordering, characterizing the functions and their requirements (performances, technical, economical) as well as constraints for product development.

As the considered electronic system is a sub-system of a larger system, it is linked to enviring entities that can be functional or physical objects. The MCSE methodology recommends following an iterative process and emphasizes on identifying the right system boundaries and its multiple life situations/contexts (or use cases) as specific configurations of the environment.

The requirements analysis approach is methodological, pluridisciplinary and creative; it's conducted within a work group involving the different stakeholders of the product.

There are many problems associated with requirements management including dealing with their changing nature. Ignoring these problems may lead to poor requirements definition, cancellation of the project or development of a system that is later judged unsatisfactory or unacceptable. Improving the requirements definition step usually results in a potentially higher quality system.

Activities for the requirements definition step	
A1.1	Definition of customer's needs and the system's purpose
A1.2	Analysis of the environment and identification of the system's boundaries
A1.3	Elicitation of <i>life situations</i> and <i>functions</i>
A1.4	Identification of functional, performances & interface requirements
A1.5	Identification of other product requirements (technical, economical, etc.)
A1.6	Identification of the system development process requirements
A1.7	Definition of test and system validation plans
A1.8	Definition of requirements and traceability management
Doc.	Requirements document

2. System Specification

Starting from customer's requirements, customer and designers have to refine initial work done together in order to clearly express the objective to be achieved and to produce a complete and verified specifications document.

The specifier uses the system requirements document as input data. Specifications are intermediate between the needs and the design and are useful to both partners. First, to the customer who can be confident that his problem is correctly understood by the designers and that the resulting product complies to his requirements. Second, to the designers who, by getting a precise idea of the problem, can efficiently undertake the design and the implementation work and better estimate development costs.

In addition, a functional specification described as an executable model can be more easily and efficiently verified by customers as it's interactively simulated or tested by computer-based tools.

This step is becoming a very important issue in system design because errors found late during product development are mainly due to a poor understanding of the requirements and the lack of specifications. To avoid those errors, designers have to remove any ambiguities, inconsistencies and incompleteness in system requirements as early as possible. Going from customer requirements to a specifications document is not as easy as it may seem and may be the most difficult step of all. Efficient specification methods have to satisfy key points:

- The specification process is based on a model or several inter-related models useful to hierarchically describe and verify system specifications.
- Specifications are described in a simple enough manner to enable comprehension, analysis and validation. This requires simple description models, graphical if possible and formal.
- The specification process guides analysts and specifiers to find, in a way as linear as possible, proper specifications to correctly represent all requirements.
- Specifications models and methods are not restricted to some specific classes of systems, nor limited to systems of small or medium complexity, so they can be widely applicable.

The purpose of the specification step is to represent a purely external view of the system, starting from needs and customer's requirements. This step requires an in-depth knowledge of the system's environment and a complete modeling of the system's behavior.

To formalize the understanding of the problem and the objective to be achieved, an intermediate document is necessary between the requirements and the solution developed by the designers (expressing the *HOW*). This intermediate description, called specifications (the *WHAT*), bridges the gap between customers and designers, and allows a formal verification by the customer of the designers' understanding of the problem. It is used to avoid the fact that what is not specified cannot be verified, and what is not verified may be wrong.

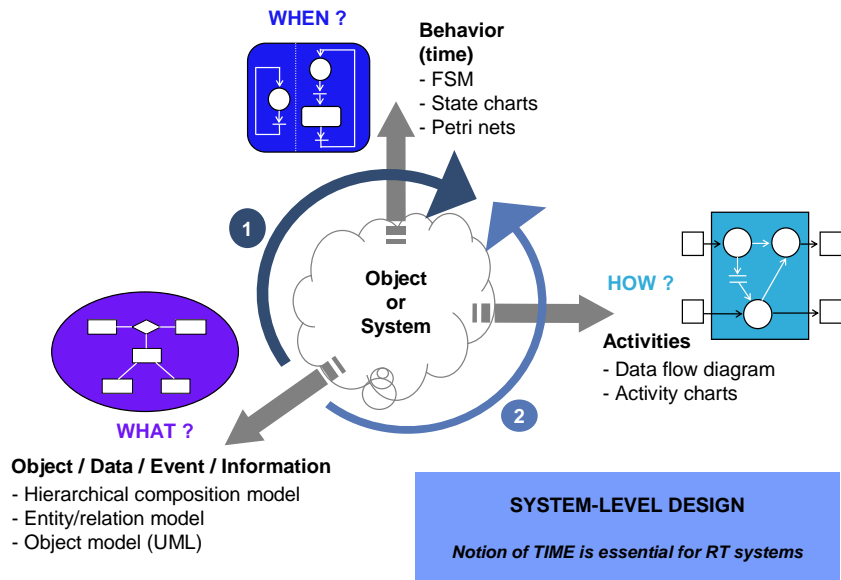
A specifications document includes **functional specifications** and **non-functional specifications**. Functional specifications describe the complete external behavior of the considered system that operates in the environment explained in the requirements document. Non-functional specifications are restrictions or constraints imposed to designers. They can be classified into product, process and external constraints. The following list gives an idea of the diversity of non-functional requirements: performances, usability, installation, interoperability, design portability, extensibility, physical and electrical constraints, environmental constraints, reliability, safety, security, availability, maintainability, costs, risks, deadline, test and validation procedures, certification, etc.

The analysis of the system's environment leads to a correct definition of the complete application and all the entities involved with the system and their inter-dependencies. The system is completely defined by its: inputs and outputs; functionality; complete and detailed behavior according to an external viewpoint; all non-functional constraints.

Due to the lack of a single model providing all required features to express full functional specifications, a coherent description of a system requires 3 complementary views:

- **Object and data modeling (*WHAT*)**
This view describes the static characteristics of each component in the system. Complex components can be refined into sub-components which structure and definition have to be described for each.

- **Activity modeling (HOW)**
This view describes the internal activities of the considered component and all its relations with the environment. This is also a structural modeling viewpoint, but from the dynamic functional features perspective.
- **Behavior modeling (WHEN)**
This view describes the temporal dependencies between the occurrence of events and the execution of actions implying data and state modifications within a dynamic component. This is equivalent to modeling the temporal evolution of objects, data or/and activities.



The 3 viewpoints for system specification

The 3 descriptions considered here are not of the same nature: static and structural (WHAT), dynamic and structural (HOW), dynamic and behavioral (WHEN). These 3 viewpoints are complementary and undissociative.

To characterize an activity, the data consumed and produced have to be defined first. To model a system, all its internal activities and their reactions to input stimuli have to be found and described. Global coherence is the result of proper descriptions for each view, which requires providing a verification method. Note that the 3-viewpoint approach implies some redundancy between the description of data and of activities, since activities modify data.

Two different modeling processes can be followed to obtain a good model having the qualities of a formal specification. The mandatory starting viewpoint is the static object and data model. But afterwards there are 2 possibilities: description of the behavior of components from the first view (path 1) or description of activities and/or their behavior separately or all together (path 2).

Methodologies differ on that point even when they are based on the same 3-viewpoint model. Traditional or functional methodologies such as SA/RT are based on path 2. Object-oriented methodologies such as UML recommend path 1. MCSE recommends path 1 when the system is not too complex; a first functional decomposition is recommended for simpler systems.

Activities for the system specification step	
A2.1	Analysis and modeling of the system environment
A2.2	Specification of all system inputs and outputs
A2.3	Description of system functions and corresponding behaviors
A2.4	Synthesis of the global functional behavior of the system
A2.5	Description of performances constraints
A2.6	Description of technological and economical constraints
Doc.	Specifications document

3. Functional Design

A system is made of components of diverse nature. An obvious category includes physical components that can be assimilated to *operators*: computers, device controller chips, memory, motors, etc. Such a description level is far from specifying the problem.

The other distinct category, which is close to the specifications level, includes logical components. A logical (or functional) component represents a set of related *operations* encapsulated within a same unit. A functional component, simply called a *function*, is really technology-independent. It characterizes functional dependencies with its environment through functional inputs and outputs (its interface) and a functional behavior, i.e. its role in relation to its environment. For example, the behavior of a component explained by an algorithm is technology-independent.

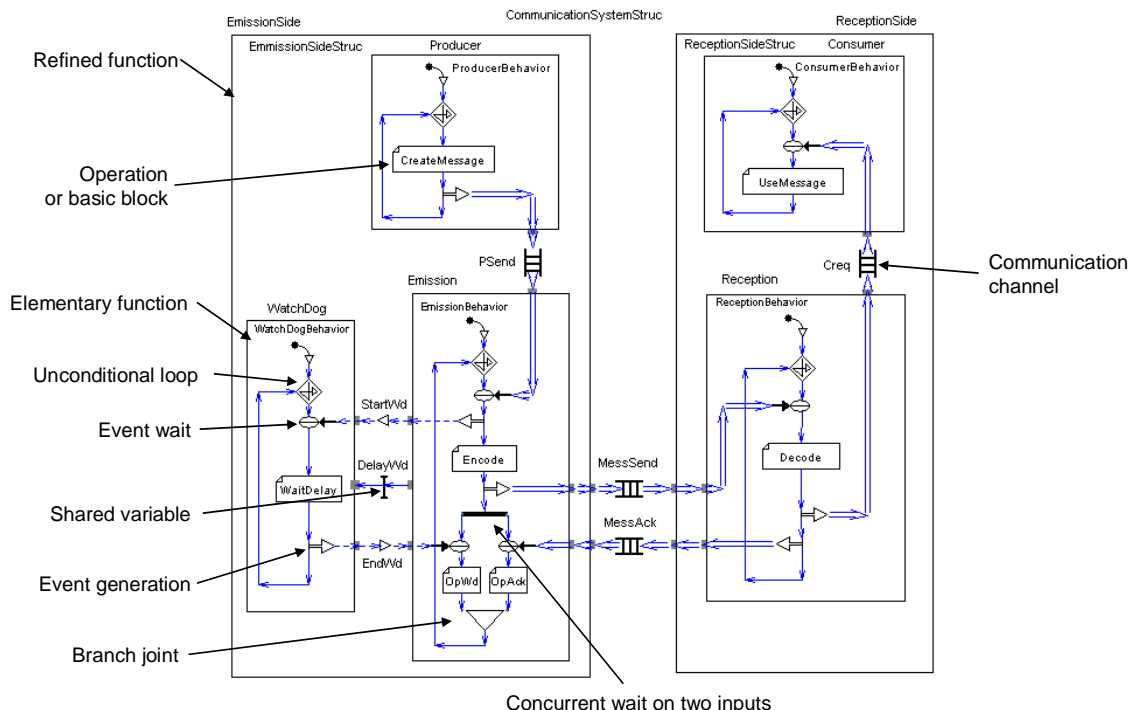
The main difference between functional and physical components is characterized by the operation-operator opposition.

A valid representation of a system is based on a structural model, representing the organization of a system decomposed into sub-sets (or functional components) and interconnections between them, and a behavioral model describing the temporal evolution of each function.

The MCSE functional model is based on the 2 complementary and orthogonal viewpoints:

- The **organizational viewpoint** (functional structure/architecture) which is described by a hierarchical structure of active functions/components and their inter-relations;
- The **behavioral viewpoint** for each leaf function (or active component) which specifies the set of operations and their complete or partial time ordering.

There is a clear difference between behavioral and structural models. Behavioral model can be refined into a structure containing simpler behaviors. Also, an abstraction operation can consist in replacing a structural part of a system by a more abstract behavioral model.



Functional model: hierarchical structure and behaviors

The purpose of the functional design step is to search for a valid functional architecture as an intermediate description between the system specifications and the organization of the physical solution. This internal

viewpoint (*HOW*), is really to be opposed to the system specifications which describe the behavior of the required system but from an external viewpoint only (*WHAT*). In the same way, the functional solution describes the "how" according to the application or problem viewpoint (function or/and operations) which is clearly different from the physical description describing the "how" for the implementation or realization (processors, operators).

The functional design defines a suitable internal architecture in an application-oriented viewpoint. The description based on a functional structure and the behavior of each function have to be technology-independent. The designer uses the functional specifications as an entry point for this step. A first functional decomposition is achieved based on the identification of the system's main internal variables (or objects) and events. Then, each function can be successively refined according to the same process until elementary functions are obtained. The behavior for such functions is defined with a purely sequential description. Data are specified as they appear during each refinement. Already designed functions from other projects can be also integrated.

During the functional design, performances and requirements are assigned to components and various trade-offs decisions are made. When the functional decomposition is complete, a global synthesis is necessary to check and obtain the coherency of the whole solution. As a matter of fact, feedbacks on previous decomposition activities can be necessary for corrections and improvements. A functional design document is the result of this global synthesis. Traceability between items in the functional solution and the specification is required.

The functional model represents a technology-independent solution and is fully executable for verifying that it behaves in conformance to the specifications. Execution times can be set to internal functions and their relations so performances can be evaluated.

The MCSE functional model is different from a specifications model (which normally describes the external behavior of the system) and also from the physical architecture. If the confusion between the 3 models is possible for small-scale problems, it is not the case when the complexity increases.

The functional model allows a smooth transition from the specifications down to the implementation and includes:

- A functional structure representing the internal organization;
- The behavior description of all elementary functions;
- The specification of all inputs/outputs, internal data and allocated constraints.

Activities for the functional design step	
A3.1	System inputs/outputs definition
A3.2	First functional decomposition
A3.3	Functional refinements and/or behavioral descriptions
A3.4	Synthesis of the complete functional solution
A3.5	Performance allocation & estimation
Doc.	Functional design document

4. Architectural Design

The architectural design step leads to expressing the specifications for the implementation of the hardware and software parts of the system. More precisely, it results in the definition of the required hardware architecture (or *platform*) and the organization of the software on each processing unit.

This result is achieved by partitioning functions between hardware and software implementation, and mapping elected functions to hardware components.

As a general trend, the software part in embedded electronic systems is increasing. An efficient development implies the early separation of the software, which is independent from the execution target, from the rest of the system. Then, the platform-independent software can be developed concurrently with the platform-dependent hardware following usual engineering methods such as object-oriented programming.

The hardware architecture design, partitioning and mapping tasks are all interdependent. Architectural issues also depend on the nature of applications: data processing, control process, signal processing, communications, etc.

The selected hardware architecture has a big influence on performances, costs, power consumption and other parameters. System architectures can vary a lot and often are heterogeneous and distributed. As architectural design concepts and tools are still emerging technologies, their choice is fundamental.

The architectural design step aims at defining the detailed solution. It consists in describing the **executive structure/support** (or hardware architecture or platform) and organizing the software on each programmable (software) processor.

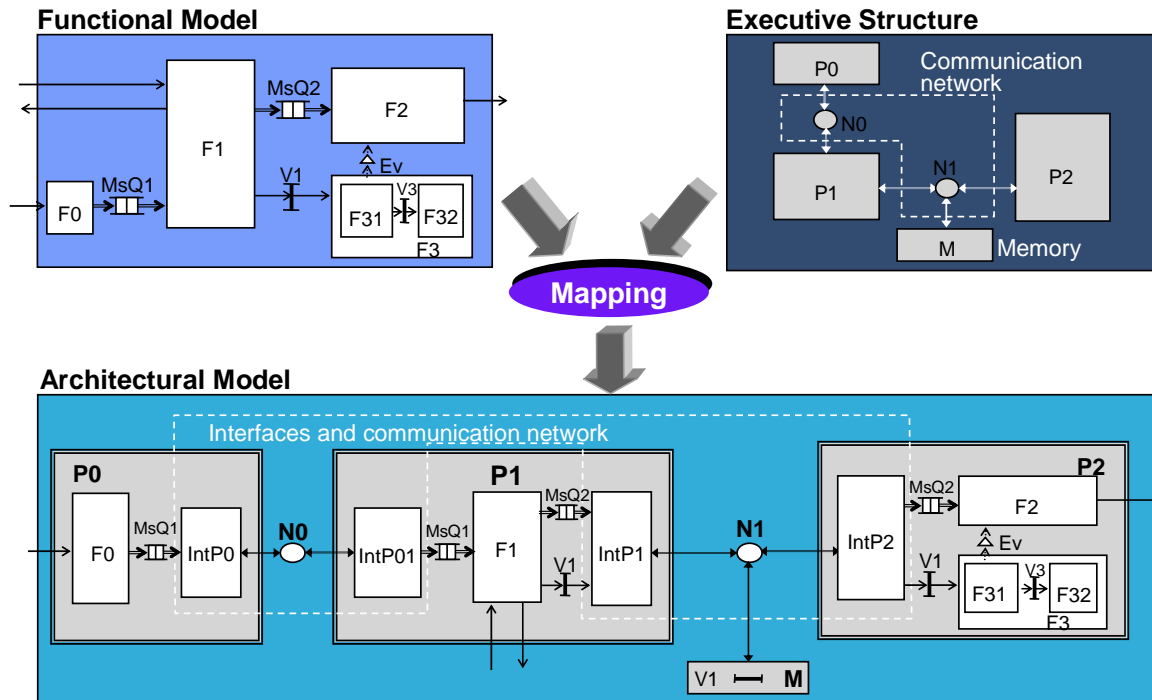
First, the functional description must be enhanced and detailed to take into account the technological constraints: geographical distribution (if applicable), physical and user interfaces. Timing constraints and performances are analyzed to determine the executive structure or physical architecture. Mapping, which includes allocating functions to software or hardware components, completely describes the implementation of the functional description on the executive structure. The behavior of the architectural solution is verified by *macroscopic co-simulation*.

The exploration of multiple executive structures and hardware/software partitioning and allocation strategies results in selection of optimal system architecture. Decision criteria include non-functional constraints such as performances, real-time constraints and costs as well as legacy components and available technologies. Such estimations are obtained by performances analysis based on a more detailed performance model.

The functional (or logical) and physical architectures represent 2 different views of the solution for a problem or system. As logical and physical elements are assimilated to and differentiated as respectively operations and operators, the physical architecture cannot be the result of a refinement of the functional architecture. It represents the set of resources used or needed to obtain the expected behavior described in the functional model. For example, a board composed of microprocessors, DSP, memories and I/O devices is a generic and programmable resource on which a wide range of functional architectures can be implemented.

A system is implemented with a set of interconnected hardware components. The executive structure is an abstract representation of the system's hardware. It is composed of programmable (or software) processors running the software programs, specific hardware (standard and programmable) components, memory components and communication links between components.

The description of an overall solution at the architectural level includes the functional view, the physical view and the mapping (or binding) between the functional components and the physical components (what operation is executed by what operator). Hardware/software partitioning consists in deciding the type of implementation (hardware or software) for each function in the functional model. The mapping defines the complete correspondence between the functional structure and the executive structure.



Partitioning, mapping and the resulting architectural model

The transition from the functional design to an architectural solution requires 4 successive operations:

- Transformation of the functional solution in order to satisfy the geographical distribution into a partitionned detailed functional solution;
- Exploration and selection of hardware architectures for the system (if the hardware is imposed, a verification of its suitability is recommended);
- Mapping of the functional model on the physical architecture;
- Specification of the implementation for the hardware and the software.

Geographical distribution represents the need to locate physical components in different areas. This is mostly the case for entities which are geographically distributed and require the system to be close to each of them. Architectural exploration takes into account performances and economical objectives. Performances and timing constraints are analyzed to determine the hardware/software partitioning.

This 4-step procedure allows the designer to efficiently determine a suitable solution that meets the required performances and minimizes the development costs.

Activities for the architectural design step	
A4.1	Geographical partitioning of the functional solution
A4.2	Addition of physical and human/machine interfaces
A4.3	Hardware/software partitioning and mapping
A4.4	Hardware & software specification
A4.5	Global verification and evaluation
Doc.	Architectural design document

5. Prototyping

The prototyping step leads to an operational system prototype. Hardware and software implementations are developed simultaneously involving specialists in both domains, thus reducing the total implementation time.

At the end of the previous architectural design step, designers elaborate a complete document describing the architectural solution, including:

- **Hardware implementation specifications** in the form of an executive structure, a model which leaves a great deal of freedom to hardware implementation specialists;
- **Software implementation specifications** for all software processors.

The software implementation specifications describe the organization of software constituents but not the programs themselves and are used as programming guide. They play an interface role between designers and implementers.

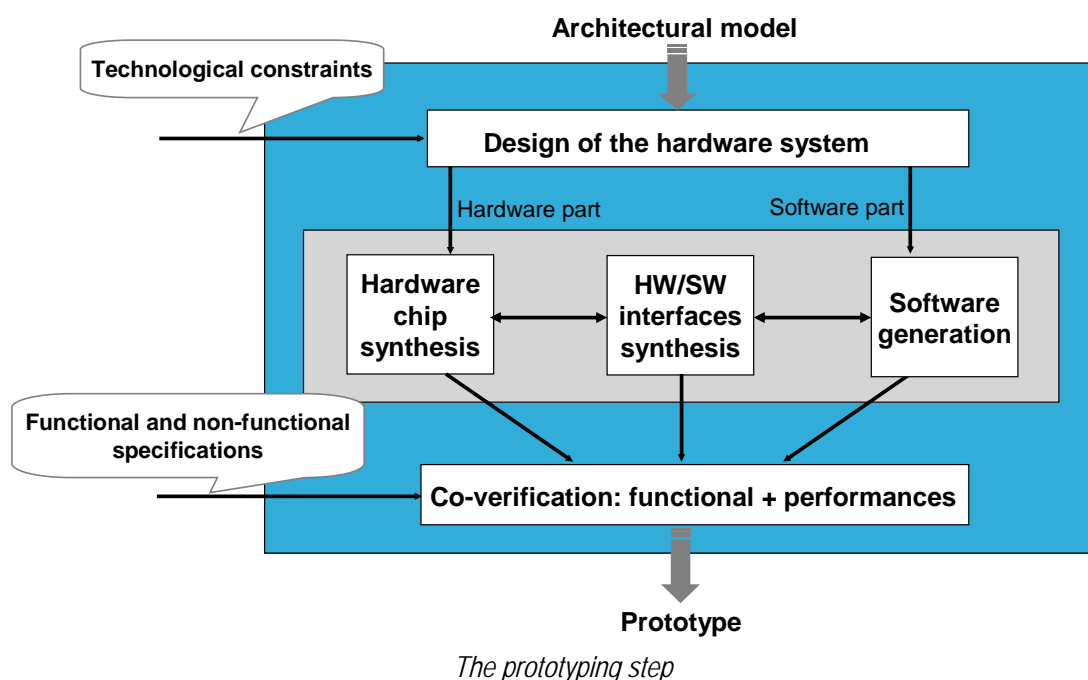
This step involves also the generation of the entire solution from the architectural design. It includes the description of the hardware architecture with the description of all the specific and/or programmable components, the software code for all microprocessors and the test and verification of the prototype.

The decomposition into the two parts – the executive structure and the software specifications – facilitates the work involved in making, integrating and testing a prototype.

The implementation activity is a bottom-up process since it consists in developing parts, assembling and testing the result, integrating the whole system in its environment and validating it with the customer.

The prototyping step is usually expensive in time and resources. Developers try to substantially reduce development time and costs by using efficient tools such as synthesizers, virtual prototypers, emulators, etc. The reuse of existing components is also a way to reduce the total time required for the project.

At this stage, an implementation strategy is chosen based on at least 3 criteria: technological specifications, techniques to be used, tools and methods available for developing the prototype. The complete solution can be generated and/or synthesized for: the hardware (ASIC and standard cores), the software and hardware/software interfaces. The resulting prototype is verified by co-simulation and emulation. Performances and other properties are analyzed and compared to the non-functional requirements.



The prototype is then transformed into a mass-produced and marketed product. Complementary industrialization criteria are introduced at this stage. Some of them have to be considered earlier such as testability after fabrication which implies design for testability.

Activities for the prototyping step	
A5.1	Detailed hardware design
A5.2	Synthesis of custom hardware parts
A5.3	Development/generation of the software
A5.4	Generation/synthesis of the hardware/software interfaces
A5.5	Functional co-verification
A5.6	Performances and constraints verification
Doc.	Prototyping document



www.cofluentdesign.com

The MCSE methodology was developed and published by the Professor Jean-Paul Calvez at the Ecole polytechnique de l'université de Nantes, University of Nantes, France (<http://www.polytech.univ-nantes.fr>).

*Books are available in English and French editions and can be **downloaded for free** at www.cofluentdesign.com:*

- *Embedded real-time systems. A specification and design methodology by Jean-Paul Calvez
Publisher: John Wiley & Sons, 1993 – 670 pages – ISBN: 0-471-93563-8*
- *Jean-Paul Calvez : Spécification et conception des systèmes. Une méthodologie
Editeur : Dunod, 1990 – 630 pages – ISBN : 2-225-82107-0*
- *Jean-Paul Calvez : Spécification et conception des systèmes. Des études de cas
Editeur : Dunod, 1993 – 270 pages – ISBN : 2-225-82230-1*
- *Jean-Paul Calvez : Spécification et conception des ASICs
Editeur : Dunod, 1993 – 580 pages – ISBN : 2-225-84216-7*

CoFluent Design Europe
24, rue Jean Duplessis
78150 Le Chesnay
France
Phone: +33 139 661 697

info@cofluentdesign.com

CoFluent Design Japan
Grace Takanawa Building 9F
2-14-17 Takanawa, Minato-ku
Tokyo 108-0074 – Japan
Phone: +81-3-4530-3868

Info-jp@cofluentdesign.com

CoFluent Design, Inc.
2033 Gateway Place, 5th Floor
San Jose, CA 95110
USA
Phone: 408-573-6172

info-us@cofluentdesign.com