# Software Model-based Development for automotive

Florent Fève, Gérard Foin, Gilles Le Calvez



# Software Model-based Development for automotive

**Authors :Florent Feve, Gérard Foin, Gilles Le Calvez**

Valeo Electronics and Connective Systems, Center of Electronics Expertise

## 1. ABSTRACT

Using modeling tools to simulate the behavior of a system before developing the corresponding product(s), is now becoming a common approach for the automotive industry, but many challenges remain. In order to obtain a seamless and efficient process for a whole software development we must pay attention to the choice of the modeling tool(s). Modeling tool(s) must be easily connected with other commercial and in-house tools used for the development. The usual software development methodology must be adapted to fully take advantage of those tools. Automatic code generation as well as modeling tools database access capability gives the opportunity to improve the global code test strategy. Whatever the performances of the tools are, the process applied remains a key point for the development.

This paper explains the innovative approach that has been developed to meet the challenges described above. The corresponding VALEO model-based development process is based on Statemate, Rhapsody in Micro C, static and dynamic test tools and in-house tools.

## 2. INTRODUCTION

It has long been recognized that ambiguous and inconsistent requirements are the primary cause of system design errors.

The global process of VALEO model-based development provides solutions to this critical issue. It involves the initial functional model, followed by the software design model, automatic code generation and the automatic software and product validation bench. In addition, the global process offers, as a spin-off, traceability between all the different development steps. The automatic link will be detailed between requirements specification and validation, as well as the automatic link between requirements specification and the code implemented in the product. Finally a general description of the current code test strategy will be given and the planned improvements will be presented.

## 3. MODEL- BASED SOFWARE DEVELOPMENT PROCESS

In recent years several modeling tools have come out. The use of these tools produces a real improvement in software development. It ensures that the product being developed relates to the goal and purpose of the system. Nevertheless, most of these tools are time-consuming, and they do not yet bring about seamless software development. Due to this lack of overall efficiency, the user cannot take full advantage of the initial time investment they require. Indeed, in many cases there is a gap between the generated model and the implementation on the one hand, and, on the other hand, between the model and the validation test of the product. After explaining these two gaps, this paper presents the solutions developed.

### 3.1 BRIDGING THE GAP IN THE IMPLEMENTATION PROCESS

The first potential gap lies between the functional reference model and further application code implementation. Using a tool to only model the functional requirements in order to have a functional reference is interesting, but one still needs to make a manual design of software and to develop the code. To avoid having a gap between the model and the implementation, automatic code generation capability is now used.

There are a lot of modeling and code generation tools on the market. Although code generation technology has significantly matured during recent years, the results are often inappropriate for embedded software. The reason is that several attempts have been made to generate embedded code from a system or functional model without making any software design. The result is that the implementation isn't suited for Design constraints such as timing constraint, tasks synchronization, timing bottlenecks, RAM and ROM constraints etc.

On the other hand, if one uses an implementation- or design-oriented modeling tool, the generated code may meet embedded software constraints (if the code generator is efficient enough), but the model will take much longer to develop and be much harder to maintain. Taking into account and evaluating the impact of change requests will be long and tedious; the risks are then similar to those taken when one goes directly from the customer requirements to the software implementation using a manual process without writing a software requirement document.

### 3.2 BRIDGING THE GAP WITH VALIDATION TESTS

The validation step of a product is a very important phase of development as to duration for two reasons. The first reason is that a great attention is required to elaborate the right tests. Indeed, due to time to market constraints, the number of tests that can possibly be executed on the product is limited. Therefore only relevant ones should appear in the Validation Plan and, as far as possible, every relevant scenario should be thought of and considered. The second reason is that tests on the product will be applied many times, from the first mock-up to the final product. Each time a new version is generated in order to take into account customer change requests, the entire validation plan will be applied for non-regression testing.

Once the model is validated it becomes the functional reference. This validation is done through simulation and record capability of the tool. When the product is to be developed, equivalent tests will be made to validate the final product. One challenge here is to compare the results of the functional simulation model with the results of the corresponding validation test on the product, since the physical characteristics of the inputs and outputs are not taken into account in the functional model. The second challenge is to reduce the duration of this validation phase by re-using the model test.

### 3.3 MODEL-BASED PROCESS

The VALEO model-based software development process has been established in order to bridge these two gaps. Figure 1 shows that, on the classical V curve, modeling and simulation tools have been added as well as an automatic code generation tool. To take full advantage of the use of these tools, internal modeling rules as well as in-house tools have been developed in order to make the process seamless and increase the efficiency of the different test steps.

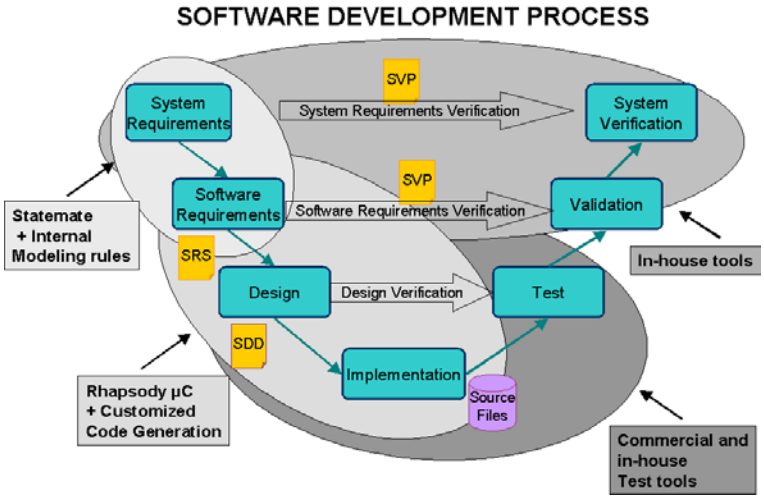


**Figure 1. Model-Based Software Development Process**

The Statemate tool is used for functional system and software modeling, and Rhapsody in Micro C tool for global software design and automatic code generation. The databases of these two tools are compatible. We clearly begin with a functional model, without taking into account the implementation constraints, to obtain a functional reference that can be validated by the customer through a PC platform executable. The model is then transformed into a global design making a new functional breakdown if necessary and adding design attributes.

The tool database access capability is used to optimize the test steps by formatting files for direct input of the code test tools.

The link between the functional model and validation is provided by an in-house executable that has been added as a plug-in in the Statemate environment. It enables the test scenarios to be captured via graphic panels and automatically generates the corresponding graphic test in a statechart format out of which test script for automatic validation bench is extracted as well as the validation plan documentation.

3.4 PROCEDURES

3.4.1 Formalization of Requirements

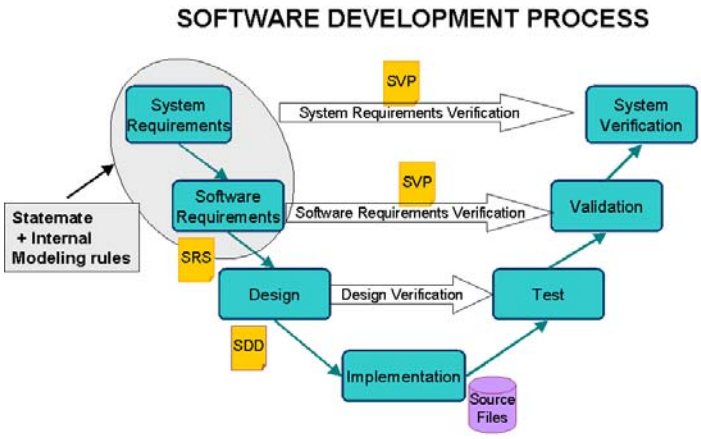


Figure 2. Formalization of Requirements (shaded area)

At the very beginning of the modeling phase ”the functional context diagram from which the tool automatically generates a graphic panel linked to the inputs and outputs of the model“ is defined. From the customer requirements the validation scenarios are very easily established through the graphical panel by clicking to enter the inputs and expected results with the exact timing.

In this very innovative part of the VALEO Model- Based Software Development Process, three elements are automatically generated (Figure 3) :

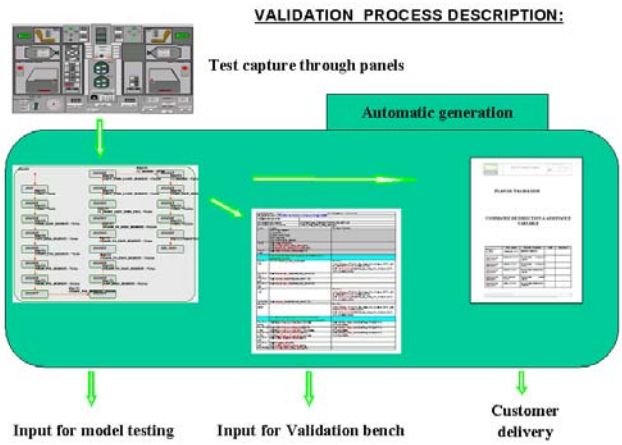


Figure 3. Validation scenarios formalization

Graphic Tests

Each captured test is generated in a state chart format, therefore each test is part of the Statemate environment and available for both model validation when developed and a non regression test when change requests occur. The state chart contains the functional sequence as well as all the necessary information for the validation sheet including the requirement being validated.

Validation Scripts

The Validation scripts used as direct input for automatic validation bench are the exact translation of the tests scenarios applied on the model.

Validation Plan Documentation

A Validation Plan Document is automatically generated including textual and/or graphical representation of each test. This validation plan is ready to deliver to the customer.

3.4.2 Requirement validation with the customer

Communication with the customer concerning the validation of the requirements understanding will be done by using :

A. Before developing the functional model

- Graphical tests (state chart format and Sequence diagram format)
- Validation Plan

B. After developing the functional model

- Sequence Diagram
- Virtual prototyping (Figure 4)
- Rapid prototyping (Figure 4)

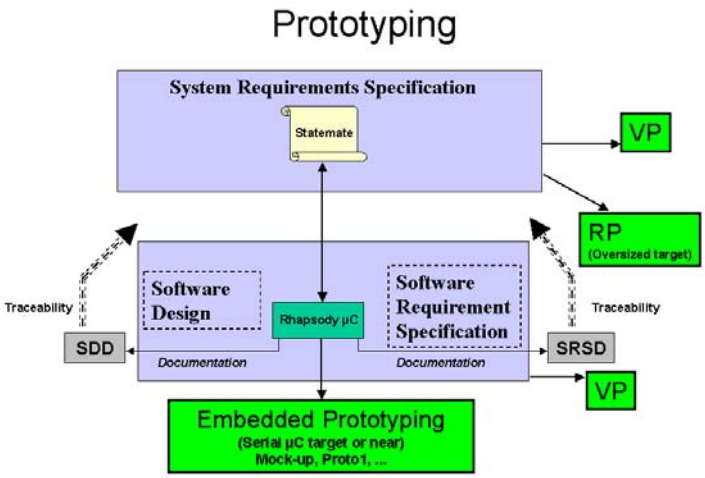


**Figure 4. Prototyping**

During model development and once finished, graphic tests through simulation test bench are applied to check the conformity. Tests can be recorded during simulation in Sequence Diagram format to be analyzed easily. In addition, a virtual prototype (PC platform executable) is delivered. Rapid prototyping on an oversized target can also be delivered in order to have a physical ECU representative of the behavior.

3.4.3 Mock-up and final product implementation

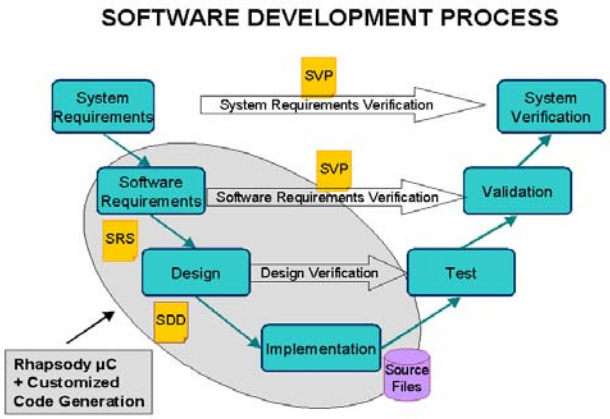


**Figure 5. Implementing the Product (shaded area)**

The challenge of the process during the downward side of the V curve is to maintain consistency, from the requirement to the implementation.

Two models are made: the first one is for the system and software functional requirements specification to validate the customer requirements, and the second for the software global design to automatically generate the code ; consequently the consistency between these two models must be checked from a block box point of view. This can be easily done because Statemate and Rhapsody in Micro C have compatible databases, and their record format files are identical.

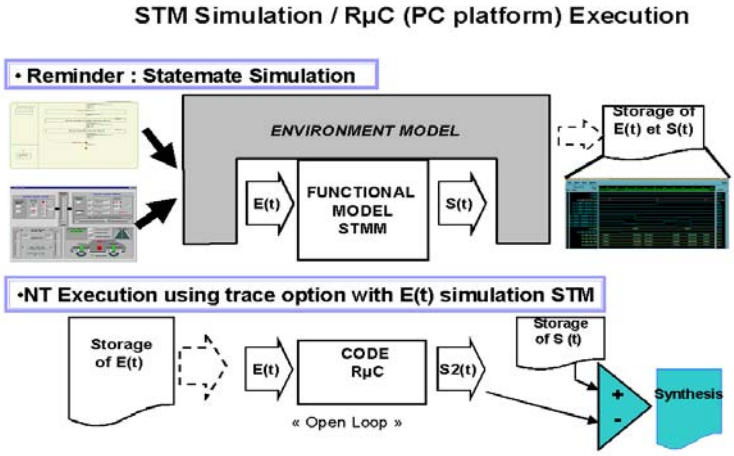


**Figure 6. Consistency check between Functional and Design model**

As illustrated in Figure 6, a black box test is performed on the functional model using the graphical tests and recording the inputs and outputs in two different files. These tests are also performed on the functional part of the

design model from which the code is generated. A PC platform executable has been generated. Input record files can be directly re-run on the PC platform executable and the results compared to ensure that the design constraints meet the requirements.

Once the consistency has been checked from a block box functional point of view, the code should be generated for a specific micro controller and linked with the low layers. Application Programming Interfaces have been defined in order to link the application layer generated automatically from the model to the low layers written manually, including local device Management, Communication management, driver components management and basic software components (Figure 7).

As to integration with OS, Rhapsody in Micro C enables the user to customize the code generation so that it will fit the OS characteristics exactly. Therefore it can target a scheduler-based implementation, as well as any OSEK or in-house OS without having to evolve the model.

Figure 7 describes the global approach for Micro controller target implementation from a model.

To have an efficient process it has been necessary to define strict modeling rules concerning model architecture and naming rules. These rules have been defined to facilitate the translation of the functional model into a design model without setting any constraint on the functional model. This one should remain readable and maintainable.

The initial step of design model development is the importation of the functional model. This model is then transformed into a design one. Task allocating, precise data typing and design attributes are defined. Then a PC platform executable (of the application part of the model) is generated in order to check the consistency between the two models.

Once the functionality is ensured, the manual C code (low layers) and any external application C code are integrated. The target configuration is also defined in a code generation profile to generate an executable for a specific target.

The design modeling tool is flexible enough to allow external C code to be linked to any part of the model. This is done either by linking a specific C code function to a specific activity (function) of the model or by linking a full C file through a code generation menu when generating the final executable.

A first mock-up is then generated including debug communication in order to test the software on the target connected to the model for graphical debug.

This first debugging and trouble-shooting of the software running on the final target are easily done through the graphical animation of the model on a PC linked to the target.

Optimization at the model level can then be made to reduce the code size. Each time an evolution is made in the design model a new PC platform executable is generated to check the consistency between the functional and design models before generating the target version.

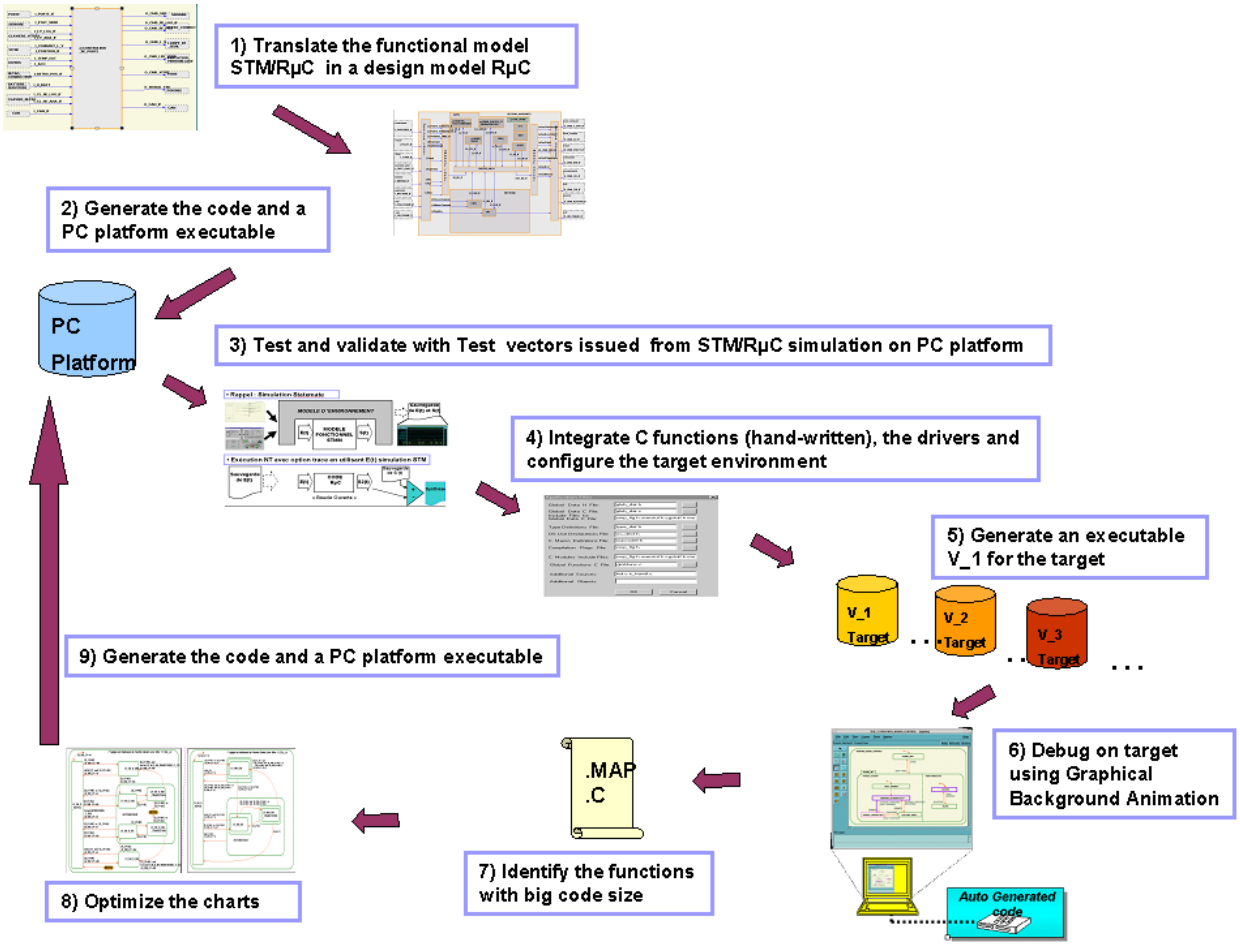


**Figure 7. Model- Based Process : global approach for implementation**

3.4.4 Implementation test

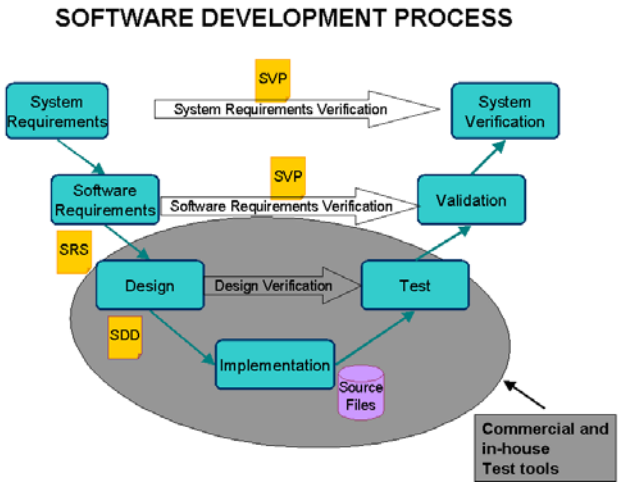


**Figure 8. Testing the implementation (shaded area)**

3.4.4.1 Static code test

Since the automatic code generator used is not certified, the same level of testing as for manual code is required. To apply the same test tools on both manual and automatic codes, these must respect the internal rules in terms of structure, presentation and naming rules. This is possible due to the fact that the automatic code generation is fully customizable.

As illustrated in Figure 9, prior to Unit Test, customized commercial tools are applied on the whole code (manual and automatic) to make static code analysis. VALEO internal naming rules compliance reports are generated and therefore, MISRA rules which are included in VALEO rules are checked.

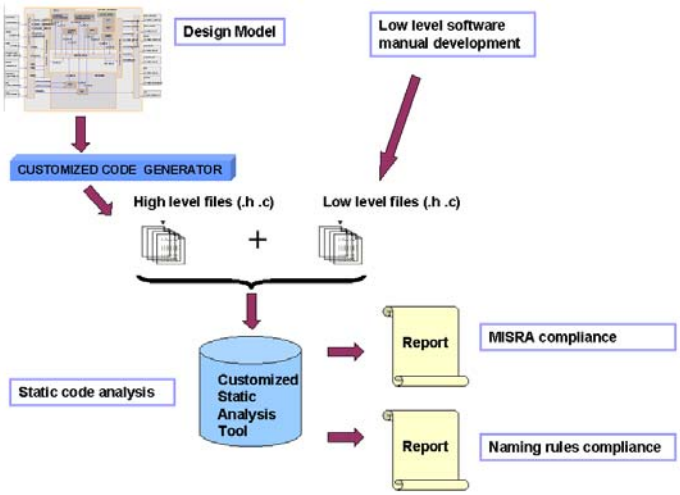


**Figure 9. Static code analysis**

3.4.4.2 Dynamic code test

Once static tests have been performed, a dynamic analysis tool is applied on the code of the complete application for run time error detection like index and arithmetic overflows.

Using the dynamic code analysis tool in combination with modeling and code generation tools significantly improves the process. Indeed one characteristic of dynamic code analysis is that the duration of the tool analysis can be rather long (from few hours to several weeks depending on the project) and that once the tool has finished the analysis, a big amount of work remains for the developers to analyze the report generated by the tool. This duration can be reduced by limiting the number of combinations. Defining a range for each input and output data will limit the overall calculation and the number of warnings, thus limiting the duration of the tool results analysis.

This range information is included, during specification and design steps, in the model data dictionary using specific customized attributes. As shown in Figure 10 the code generator is customized so that, in the generated C files, the data range appears as a comment which is identified with a specific token. An in-house tool automatically extracts the range information and configures the C files so that the dynamic code test tool optimizes the analysis.

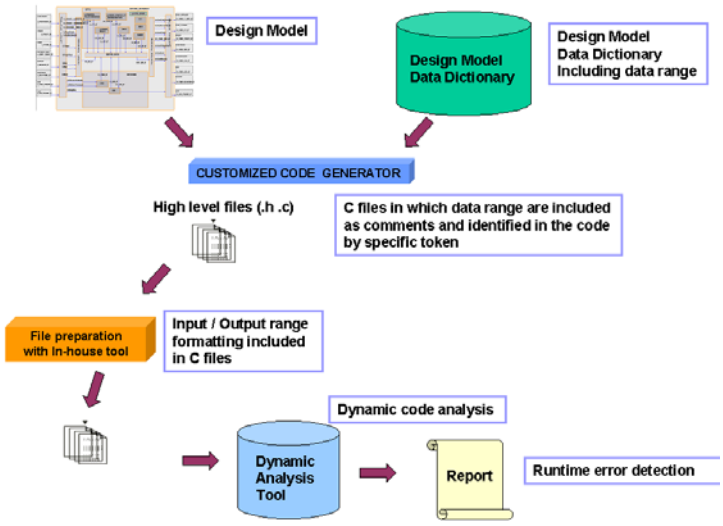


**Figure 10. Run time error check**

3.4.5 Product validation

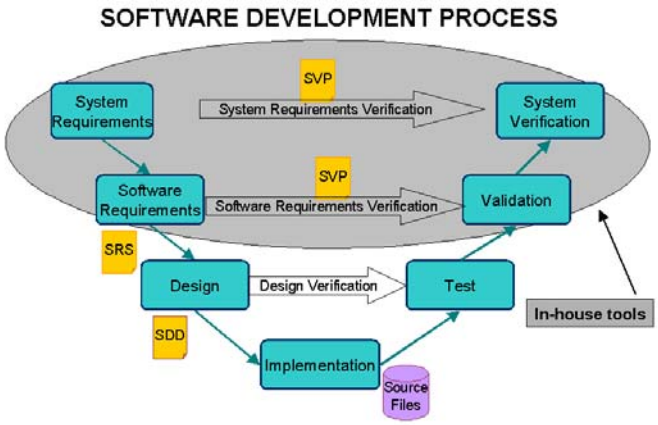


**Figure 11. Product Validation (shaded area)**

As mentioned above, Validation scripts are used as a direct input for automatic validation bench. They are the exact transcription of the test scenario applied on the model. This is a great improvement compared with the process in which model testing is first recorded and then equivalent scenarios are handwritten in a specific script format to rerun the same test on the final product to compare the results.

Extracted from the test state charts, the test scripts contain the functional sequence as well as all necessary information for the validation sheet, including the test number, the description of the test, requirement(s) under validation and validation bench configuration information if necessary. The scripts are generated in RTF format with a generic format for stimulating inputs and verifying outputs.

Macros are used to translate the functional inputs and outputs into physical ones, taking into account the physical characteristics of each ECU and allowing very easy adaptation in case of hardware changes. Whatever the physical support of the inputs/outputs may be (logical, analogical, PWM, network), the validation plan and validation script will be up-to-date with this methodology. Only macro implementation takes into account the physical support.

Once the scripts have been executed on the automatic Validation Bench, the results are compared with the simulation results which are the reference for the validation tests (Figure 12).

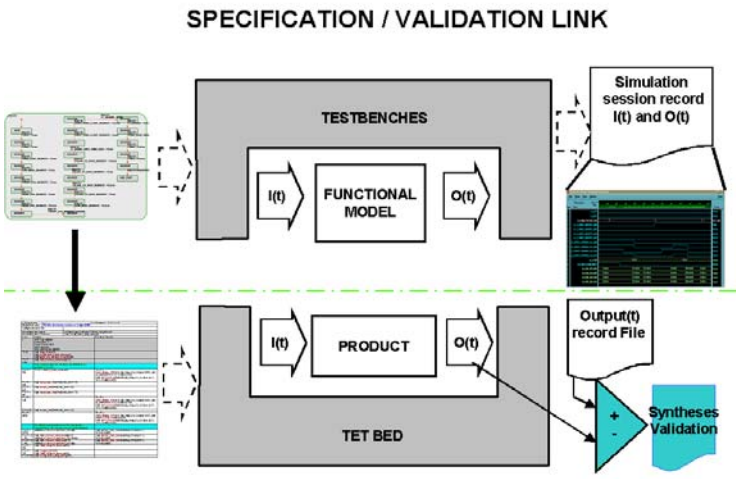


**Figure 12 Validation Process**

The global approach for model and product validation is described in Figure 13.

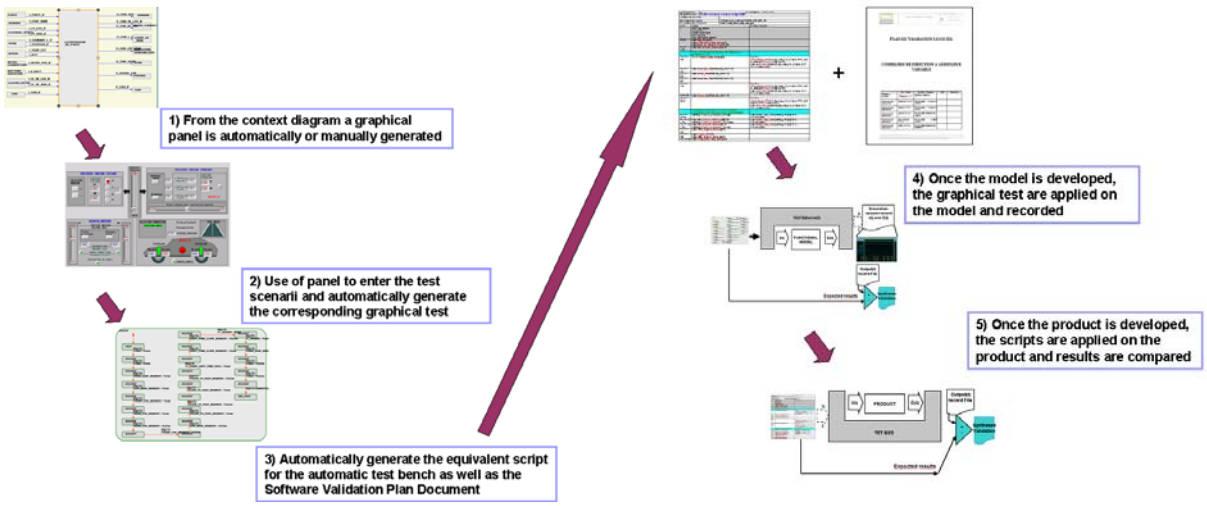


**Figure 13. Model-Based Process : global approach to validation**

3.4.6 Traceability management

Another great item of interest of this process, is the traceability capability.

An Attributes-specific field of the modeling tool allows the user to enter pre-formatted information linked to any element of the model. This Attributes field is used to establish the traceability.

At the beginning of the development, textual requirements are analyzed and each requirement is identified. A label is assigned to each requirement.

- The reference to the requirement(s) is established during the definition of test scenarios via graphic panels. Automatic generation of the graphical test fills the Attributes traceability field with the requirement(s) label(s).
- When developing the functional model each element implements one or several requirement(s) . Again traceability is ensured by filling the Attributes traceability field with the corresponding label(s).
- The design inherits this traceability automatically since the functional model is the starting point for the design model . The Attributes traceability field is filled for each added design element.
- Finally the code generation profile has been customized so that traceability information (label(s) of the Attributes traceability field) appears as comments for every C code function.

An in-house development enables users to extract the total traceability matrix in an Excel format through a fully integrated tool suite (Figure 14).

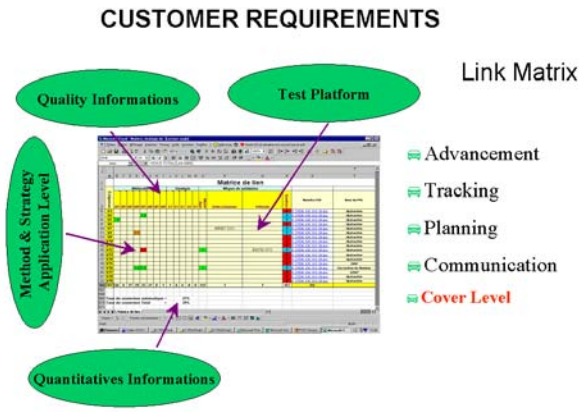


**Figure 14. Traceability Matrix**

When a change request is made, the impact on the functional model, software design, C code functions, and validation test is immediately identified.

## CONCLUSION

A seamless model based development process has been demonstrated and its efficiency validated. It has actually been shown to improve communication between different members of the development team and customers, and to reduce time to market. This has been achieved while increasing the overall development quality.

Obviously, the availability of functional system models provides a safe ground for all the following software development and validation. The entire Valeo process takes full advantage of this ground in a seamless development, with tools which are all connected to this initial step. The gaps often observed in validated development methods, as mentioned above, have been bridged.

This work has been made possible, because the tools used are open tools. The accessibility of the modeling tools database allows efficient process customization to our goals.

This innovative process has been applied partly to the development of an industrial project currently under production. It has already brought great quality improvement and significantly reduced development time. Nevertheless the process can be improved in particular for C code testing. Work is already planned to take advantage of the code generation combined with data range information to optimize the code Unit tests. The aim is to automate the unit tests to have a coverage analysis as well as functional, structural and robustness run time error detection.

The automatic link between specification and validation is a key in the process ; it dramatically reduces the validation phase, ensures the consistency of the process and is very flexible. It can be very easily adapted to any specific automatic test bench modifying a small and well-identified part of the in-house tool.

Regarding validation, new tools based on mathematical analysis of models such as BDD or Symbolic representation, where the goal is to generate automatically relevant tests from a model, are also being evaluated. The current process is ready to integrate these tools as soon as they have been fully developed.

## ACKNOWLEDGMENTS

The authors thank P. Germanicus, G.M. Martin and S. Zamia for advice and support.

## CONTACT

Florent Feve, CEE Software Expertise Engineer.
11 years of experience in Automotive Software Development, joined VALEO Center of Electronics Expertise in 2000 as modeling and code generation tools specialist.

Gérard Foin, CEE System dependability department manager.
33 years of experience in Aeronautics and Automotive Electronics Development, joined VALEO as Manager of Software Department of the CEE in 2001 and System dependability department manager since 2003.

Gilles Le Calvez, Director of VALEO Center of Electronics Expertise.
14 years of experience in Aeronautics and Automotive Software Development, joined VALEO as Manager of Software Department of the CEE in 1997 and Director since 2001.

Valeo Electronics and Connective System
2, avenue Fernand Pouillon
Europarc
94042 Creteil Cedex – France
Tel. : 33 1 45 13 78 12
FAX : 33 1 45 13 78 68
florent.feve@valeo.com

## DEFINITIONS, ACRONYMS, ABBREVIATIONS

**BDD**: Boolean Diagram Decision

**ECU**: Electronic Control Unit

**RP**: Rapid Prototyping

**VP**: Virtual Prototyping