Teaching Model-Based Systems Engineering with MATLAB & Simulink

line 1: 1st Given Name Surname   
line 2: *dept. name of organization   
(of Affiliation)*  
line 3: *name of organization   
(of Affiliation)*line 4: City, Country  
line 5: email address or ORCID

line 1: 2nd Given Name Surname  
line 2: *dept. name of organization   
(of Affiliation)*  
line 3: *name of organization   
(of Affiliation)*line 4: City, Country  
line 5: email address or ORCID

line 1: 3rd Given Name Surname  
line 2: *dept. name of organization   
(of Affiliation)*  
line 3: *name of organization   
(of Affiliation)*line 4: City, Country  
line 5: email address or ORCID

*Abstract*— The development of complex energy systems is a challenge for today's industry. Systems are developed consisting of power hardware, control and communication hardware, software of all kinds and with more and more connectivity. Typically, engineering students at Master's level specialise in technologies to later apply them during their career. However, in terms of organisation and processes, a company is far from what students see in class. The development of a product implies several phases, from customer requirements definition to systematic verification and validation. Mondragon Unibertsitatea has identified the need to work with students on the whole life cycle of a product, not only in the design process as usual. This paper presents the development of a curriculum to work on Model-Based Systems Engineering with Matlab & Simulink. The main objective was to generate educational resources so students can focus on different phases of the life cycle, such as requirements definition, architecture design and management and continuous verification and testing. In this way they are trained on tasks they will find during their professional career as validation engineers, sales engineers or post-sales engineers.

Keywords— Model-Based Systems Engineering, energy systems, requirements, architecture, validation, verification, testing, v-model, life cycle management.

# Introduction

Nowadays, the development and manufacturing of a product has several phases. In the field of energy systems, complex products consisting of many subsystems, which can be both hardware and software, are often commercialized. In addition, today the connectivity brought by Industry 4.0 makes today's systems cyber-physical systems. A set of mechanisms is controlled by computer-based algorithms.

In this context, it is necessary to establish a methodology that enables the development taking into account the principles of RAMS (Reliability, Availability, Maintainability and Safety). Today, one of the most widely used standard processes in this regard is the so-called V-model (Figure 1).

Although originally intended for software development, it is now being extended to systems engineering. Today it has been adopted by governments such as the German [1] or the United States for the development of transportation systems [2]. It has also been standardized by IEC 62278 for the railway industry and ISO 26262 for the automotive industry.

|  |
| --- |
| Figure 1 Electric scooter component architecture diagram |

Moreover until a few years ago, the development process of cyber physical systems was a document-centric methodology. The information generated in the process was collected and transmitted by means of documents. This way of working has become increasingly difficult to manage, given the complexity of today's systems. It is difficult to represent all the viewpoints from which a system can be looked at through documents and to keep them updated as the design and the life cycle progresses. In response to these difficulties, companies have adopted a new way of working called Model-Based Systems Engineering (MBSE). In the MBSE philosophy, a single virtual model represents the requirements, architecture and behaviour of a system. In the field of energy systems, MATLAB & Simulink are the main tools to model and simulate products under development. Apart from being used for model-based design and simulation, in recent years, this software has incorporated functionalities related to systems engineering, such as requirements management (Requirements Toolbox), architecture management (System Composer) or systematic testing (Simulink Test).

In this context, this article presents a curriculum development project that aims to bring MBSE into an engineering classroom. Specifically, it presents the design of different activities carried out for the Master's Degree in Smart Energy Systems at Faculty Engineering of Mondragon Unibertsitatea. Through this curriculum, the student is introduced to the application of the V-model processes using MathWorks tools. The work focuses on three phases of the V-model. First, on how to perform the requirements management (Phase 3) with the Requirements Editor. Second, on the functional and formal decomposition (Phase 4) using System Composer. Finally, on the validation against the requirements using Simulink Test (Phases 7 – 8).

The learning outcomes of this curriculum are:

* Manage the life cycle of energy systems with the V-model and model-based techniques.
* Write and manage requirements with Matlab Requirements Editor.
* Compose system architectures and organize system components with System Composer.
* Arrange verification and validation tests against system requirements with test harnesses and test suites.
* Recognize the ability of MBSE to facilitate traceability between requirements, architectures, designs and verification and test results.

In addition, the application of the V-model for an electric scooter embedded traction control is used as a practical example in the classroom activities. Figure 2 shows a block diagram of an electric scooter.

|  |
| --- |
| Figure 2 Electric scooter component architecture diagram |

The structure of the article is as follows. Section II gives a brief introduction about the V-model. Section III shows the basics about requirements and their management in Requirements Editor. Section IV presents a candidate architecture for the electric scooter and the definition of different components in System Composer. In section V the way to link behavioural models with architecture models is presented. Following the phases of the V-model, section VI explains the use of Test Harnesses and the Test Manager to manage verification, validation and testing. Finally, section VII presents the conclusions.

# V-Model

As it was previously mentioned, the V-model is a standard methodology for the management of the life cycle. The model is divided into two branches. The left branch contains the processes of conception, requirements definition and design (high-level and detailed). This process leads to product development and manufacturing (hardware and software). In the right branch, the integration, verification and validation processes are performed. In this model the time axis is bended to form a V and put each phase on the left branch in the same level as its counterpart in the right branch.

The left branch is composed of the following phases:

* Phase 1: Customer requirements.
* Phase 2: Conception. The needs of the main customers and the operating environment of the system to be designed are identified and documented in collaboration with the customer.
* Phase 3: System Requirements. Customer requirements are translated into system requirements. What the customer is looking for is interpreted and translated into technical requirements to be followed during the design of the system.
* Phase 4: System Architecture. A high-level architecture is defined in order to fulfil system requirements. System boundaries, formal and functional decompositions are performed and subsystems and interfaces identified. Design of the high-level architecture of the system and mapping of subsystems with the requirements.
* Phase 5: Detailed design. Design of subsystems and components.
* Phase 6: Implementation, Software and Hardware Development. Select appropriate technology and develop software and hardware to meet component-level requirements.

In these first six phases the system is decomposed starting from the high-level architecture. In the following phases, verification and validation activities are performed while integrating all the subsystem:

* Phase 7: Unit testing. Test each hardware and software component, verifying their correct operation at unit level.
* Phase 8: Subsystem integration: Integrate software and hardware components to create subsystems verifying their correct operation at subsystem level.
* Phase 9: System testing and validation. All subsystems are integrated to form the final system. System tests are performed and validation is done against system requirements.
* Phase 10: Operation and maintenance.
* Phase 11: Upgrade or retirement.

During this process, system documentation is created. For each phase on the left side, the requirements that guide the next phase are written, as well as the validation plan for the equivalent level on the right side. For each phase on the verification and validation side, documentation for user training and validation is created.

It is important to state the difference between verification and validation activities. On the one hand, verification activities are defined as the assessment that a product or service complies with design standards or specifications. During this process, the following question is answered: Are we developing the product correctly? In short, it is about ensuring that the system built is well designed, safe and functions correctly. This process evaluates against internal requirements. Verification involves only one phase or two consecutive phases.

On the other hand, validation is defined as the assessment that a product meets the customer's requirements and needs. It usually involves external development stakeholders. During this process the following question is answered: are we developing the right product? Validation is a relatively subjective process that evaluates how well the product solves the customer's problem. That is why system validation is done against system requirements.

# System Requirements Specification

Once customer requirements are received and a concept is proposed, it is time to write system requirements. In [IEEE-STD-1220-1998] a requirement is defined as a statement that identifies a product or process operational, functional, or design characteristic or constraint, which is unambiguous, testable or measurable, and necessary for product or process acceptability (by consumers or internal quality assurance guidelines).

With regard to the curriculum, the main objective of the requirements module is to convey to students that requirements are a tool for communication between people. As stated before, nowadays systems are more and more complex and their design, development and validation should have requirements as reference.

Requirements should have the characteristics shown in Table 1.

Table 1 Characteristics of a requirement [3], [4]

|  |  |
| --- | --- |
| **Characteristic** | **Description** |
| **Feasible** | Technically possible (cost and schedule). |
| **Verifiable** | Each statement can be tested. There is a feasible procedure to do so. |
| **Unambiguous** | Each statement is precise, there are no ambiguities. There is only one interpretation. |
| **Clear** | Each statement can be understood. There are not spelling or other kind of mistakes. |
| **Atomic** | The requirement defines a single traceable element. |
| **Legal** | Does not make you break any law. |
| **Abstract** | The requirement does not specify a certain solution. It may be difficult in projects where a solution was already developed. |
| **Complete** | It contains all the information to understand it. |

The IEEE Standards Style Manual [5] recommends to use the word *shall* to indicate mandatory requirements strictly to be followed. The word *should* indicate that among several possibilities, one is recommended without excluding others. Table 2 gives some examples about requirements writing.

Table 2 Requirement quality examples

|  |  |  |
| --- | --- | --- |
| **Requirement quality** | **Requirement** | **Characteristic** |
| Poor | The system should give full power at any speed. | Feasibility |
| High | The system shall give full power up to 2000 rpm. | Feasibility |
| Poor | An amplifier stage must amplify the input voltage by a factor of 2. | Abstract |
| High | The input voltage shall be amplified by a factor of 2. | Abstract |

In MATLAB, requirements are managed using the Requirements Editor from the Requirements Toolbox. This is a tool that enables authoring and organizing requirements but most importantly, linking them with other artifacts developed in MATLAB & Simulink.

Figure 3 shows the summary table of the system requirements defined for the electric scooter. The main attributes of the requirements are shown. In addition, the implementation and the verification level are also shown. Once an architecture has been defined in Simulink (see section III), each requirement can be linked to a specific component. Depending on the existing links, the editor sets a level of implementation of the requirement. In addition, the verification status is obtained from the tests results programmed using Simulink Test (see section VI).

|  |
| --- |
| Figure 3 Scooter system requirements defined in Requirements Editor |

Each requirement has an attached sheet where all its information is entered (see Figure 4). One can define three types of requirements. The container type requirement enables grouping requirements. It is equivalent to a heading in a document (see requirement #9 in Figure 3). Functional requirements are the core type and describe the expected behaviour of the system. This kind of requirements can be linked to components and Requirements Toolbox obtains implementation and verification status for them (see requirement #17 in Figure 3). Finally, one can define informational requirements (as #53), non-functional or additional information required by the designer. These requirements are not included in the calculation of implementation and verification status.

One of the most interesting functionalities of this tool is the possibility to link requirements among them or to other artifacts in the project, such as components in Simulink or tests. For example, Figure 4 shows links to another requirement in the same set and different components in the architecture.

|  |
| --- |
| Figure 4 Requirement sheet |

Finally, it is worth mentioning that Requirements Editor incorporates analysis tools that make it easier to understand the relation between requirements and other artifacts. Figure 5 presents a traceability diagram generated automatically that shows the link between several requirements.

|  |
| --- |
| Figure 5 Traceability diagram for requirement #53 |

The next section shows how to build a system architecture using System Composer and how to link requirements to that architecture.

# SYSTEM ARCHITECTURE DESIGN

Phase 4 in the V-model life cycle is system architecture design. Once customer requirements are clear, a concept is proposed and system requirements specified, it is time to propose an architecture that meets client's needs. The main tool for that in the MATLAB & Simulink environment is System Composer.

Figure 6 shows the formal architecture of the electric scooter. It was designed following the system requirements presented in section III. The formal architecture represents the composition of the system with all the components and the signals they exchange. With System Composer the design can be built in the Simulink environment and linked to information stored in other MATLAB & Simulink tools. The following subsections show the different functionalities of this tool.

|  |
| --- |
| Figure 6 Formal architecture design in System Composer |

## Architecture design

The basic element to create an architecture is the component. Components are boxes that can be filed with more components or simulation behaviour models. As shown in Figure 6, components can have data inputs and outputs, but physical interfaces (diamond shaped) can also be used. With physical inputs, Simscape models can be integrated in the architecture.

## Requirements linking

Each component, or even the whole architecture can be linked to requirements defined in the Requirements Editor. This helps maintaining traceability and checking the consistency of the proposed architecture. The requirements view of System Composer and Simulink shows the mapping between requirements and components. Moreover, the Editor shows which components implement which requirements and the implementation status. With this feature, students can start understanding the fundamentals of Model-Based Systems Engineering. They will start using a model centric approach that gathers components and requirements in the same environment.

## Stereotypes

Stereotypes are used to define component or signal types so the architecture can be organized and analysed more easily. For example, in the scooter architecture, software and hardware components have been defined. Within the software components there may be some dedicated to traction control, others dedicated to communications and others to the management of hardware inputs and outputs. By defining stereotypes, the components can be classified and facilitate development management. In the architecture shown in Figure 6, the components were classified in the following groups: storage systems, power electronics, sensing, electric machine, mechanical parts, actuators and software. Moreover, each stereotype has its own attributes, such as mass, nominal power or useful life. These attributes may be used to launch different analysis about the feasibility of the proposed solution and compare different alternatives [6].

## Views

Once the architecture is defined and stereotypes applied, views help analysing the structure from different points of view. System Composer has integrated an Architecture Views gallery where component hierarchies, architecture hierarchies and sequence diagrams can be created interactively.

In Figure 9 a component hierarchy view can be seen, where a filter was applied to only see the power electronics components. This tool is very useful for developers when they want to focus in a specific part of the design. Views are created interactively but the complete information of the system is still present in the model. Compared to a document centric approach, with this model-based tool there is no need to create ad-hoc documents about the architecture for each team involved in the design.

|  |
| --- |
| Figure 7 Embedded traction control strategy for electric scooter |

# Linking Detailed Designs with Architectures

In principle, it is not the objective of this curriculum to work on the detailed design of embedded controls for electric drives. It is assumed that students have previously acquired sufficient knowledge to understand the hardware and software components that make up an electric scooter. However, it is worth mentioning that architectures designed with System Composer can be linked with dynamic behaviour designs developed in Simulink.

The control structure used for the example is shown in Figure 8. It is composed by a cascade structure that controls speed and/or current/torque of the motor according to the reference established by the user in the throttle.

|  |
| --- |
| Figure 8 Embedded traction control strategy for electric scooter |

Within the curriculum activities, it is proposed that students merge the components of the architecture with the dynamic model. Thus, if the architecture includes the links to the requirements, an executable specification is available in a single file for continuous verification and validation.

Figure 9 shows the behaviour model of the current controller linked to its architecture component. Since the architecture component is linked to some requirements, the link section in the Requirements Editor is updated to show it. Moreover, as it is presented in Figure 3 a column showing the implementation status also appears.

|  |
| --- |
| (a) |
| (b) |
| Figure 9 Behaviour simulations linked to architecture components (a) architecture component (b) behaviour model inside torqueControl component |

A full electric scooter model developed with System Composer and Simulink lets designers to perform continuous verification and validation (V&V) activities during the whole life cycle, not only in the late phases of the V-model. Figure X shows some simulation results performed during the tuning of the control strategy. It is worth mentioning that the tunning of the controllers was also carried out with the Control System Toolbox of MATLAB.

|  |
| --- |
| Figure 10 Current control simulation results |

From these results, control code programming would be the next step. The code may be generated manually or using automatic code generation tools, such as Simulink Embedded Coder. This task is out of the scope of the curriculum, so the verification and validation processes will be explained in the simulation environment shown up to now.

# Validation and Testing

The phases of unit testing, subsystem integration and system validation follow a bottom-up strategy. First, the correct behaviour of atomic components is tested. After, two or more components are integrated and tested. Finally, a general system validation is performed against system requirements. The process follows the grouping of components selected in the architecture. Simulink Test helps to manage this process with test harnesses and Test Manager.

## Test harnesses and unit tests

Unit testing focuses on isolating atomic components from the general design and testing their behaviour in standalone mode. In order to that in MATLAB & Simulink each component can be isolated using test harnesses. A test harness is a separate model file where the component is tested and all the artifacts needed for that are included. Figure 11 shows a test harness for the current control mentioned before.

|  |
| --- |
| Figure 11 Current control test harness |

As it can be seen, the harness is embedded in the main simulation file and is linked to its parent component. Simulink Test handles any change made to the component, so all the instances are updated continuously. The harness shown in Figure 11 includes a test sequence and a tests assessment block. These enable to implement test inputs and verification code programmatically with MATLAB language. If the component under test needs any additional block to check its operation, it can be added here. In the example, the current loop uses a simplified transfer function of the motor to test its response. In this way there is no need to run all the components in the architecture, so unit testing is simplified.

Apart from the test sequence and test assessment blocks, several alternatives can be used to configure inputs and check outputs, such as Signal Builders, charts or signals imported from the workspace. Finally, it is worth mentioning that test harnesses can also be stored in a separate file, making it easier to distribute tasks in validation teams.

## Test manager

The general philosophy of Model-Based V&V is that the sooner designs can be verified, the sooner bugs will be detected and corrected. If the system is small enough, test harnesses, tests and results may be handled manually. However, in complex systems, this can lead to a large number of tests and results. The example used in this curriculum project, in a first approximation of system requirements (top level requirements), already has 43 of them. If it is supposed that each subsystem or component has its own requirements, testing management may be a challenge.

Simulink Test Manager is the main tool in MATLAB to author, manage and execute Model-Based testing. Once test harnesses and environments are configured for a design, Simulink Test enables executing all of them in batch and getting the results in the same interface.

As it is presented in Figure 12 tests are grouped by test files, suites and cases. There are different test case types. In the one shown below, a simulation test was configured to execute the harness presented in section VI.A. Baseline tests can also be configured, where the results of a simulation are compared to baseline data, for example results from a previous simulation or laboratory results. Equivalence tests are used to compare two simulation results.

|  |
| --- |
| Figure 12 Test Manager and test case configuration |

Once tests are configured, the Test Manager executes them in batch. Simulations are run in the background and results stored in the test file. Each test case can be linked to requirements defined previously in the Requirements Editor, which shows the verification status depending on the results of the test cases. Thanks to this feature, students have a powerful resource to understand how test cases are linked to requirements, how Model-Based tools can help validating designs and how traceability tools can help manage the development of complex systems.

As an example, Figure 13 shows the results of a test where the steady state error of the current is checked. This is the result of simulating the test harness shown in Figure 11. Apart from simulation signals, Simulink Test shows sample by sample the result of the verification test (in green samples that passed, in red samples that did not and in grey samples that where not checked).

|  |
| --- |
| Figure 13 Test Manager and test results |

# Conclusions

##### This article shows the development of a curriculum for working with MBSE in engineering studies. Based on the V-model for life cycle management and MATLAB & Simulink, the necessary resources have been created to work on the most important aspects of this field. The main learning outcome for the students was to learn manage the life cycle of energy systems with the V-model and model-based techniques Firstly, it has been shown how to create a set of requirements in the Requirements Editor following the guidelines established in the IEEE Standards Style Manual. Then, the creation of architectures and the linking with requirements and dynamic behaviour models was presented. Finally, the process and tools for verification, validation and testing against requirements were presented.

As a main conclusion it can be said that the tools integrated in MATLAB & Simulink can be a suitable starting point to start working on MBSE in the classroom. In the field of energy and electronics, academic institutions usually use this software in the classroom. If students are familiar with it, they can start working on systems engineering concepts without the need for specific training.

##### Acknowledgment

The preferred spelling of the word “acknowledgment” in America is without an “e” after the “g”. Avoid the stilted expression “one of us (R. B. G.) thanks ...”. Instead, try “R. B. G. thanks...”. Put sponsor acknowledgments in the unnumbered footnote on the first page.

##### References

[1] M. Meisinger and I. H. Krüger, “A Service-Oriented Extension of the V-Modell XT \*,” 2007.

[2] US Department of Transportation, “Systems engineering for intelligent transportation systems,” p. 11, 2007, [Online]. Available: http://ops.fhwa.dot.gov/publications/seitsguide/seguide.pdf

[3] E. Hull, K. Jackson, and J. Dick, *Requirements Engineering*. London: Springer London, 2011. doi: 10.1007/978-1-84996-405-0.

[4] I. C. S. Committee, “IEEE Guide to Software Requirements Specifications,” IEEE, 1984.

[5] IEEE Standards Association, “2021 IEEE SA Standards Style Manual,” 2021. Accessed: Jan. 02, 2023. [Online]. Available: https://mentor.ieee.org/myproject/Public/mytools/draft/styleman.pdf

[6] T. M. Inc, “Analyze Architecture Models.” https://www.mathworks.com/help/systemcomposer/analyze-architecture-model.html (accessed Feb. 21, 2023).