Teaching Model-Based Systems Engineering with MATLAB & Simulink for Smart Energy Systems

Jon del-Olmo Larrañaga
Mondragon Unibertsitatea
Faculty of Engineering
20500 Arrasate-Mondragon, Gipuzkoa, Spain
jdelolmo@mondragon.edu

Iosu Aizpuru Larrañaga
Mondragon Unibertsitatea
Faculty of Engineering
20500 Arrasate-Mondragon, Gipuzkoa, Spain
iaizpuru@mondragon.edu

Manu Sanchez Alberdi
Mondragon Unibertsitatea
Faculty of Engineering
20500 Arrasate-Mondragon, Gipuzkoa, Spain
manu.sanchez@alumni.mondragon.edu

Jennifer J. Gago Muñoz
MathWorks
Madrid, Spain
jgagomu@mathworks.com

David Gonzalez-Jimenez
Mondragon Unibertsitatea
Faculty of Engineering
20500 Arrasate-Mondragon, Gipuzkoa, Spain
dgonzalez@mondragon.edu

***Abstract*— Developing complex and smart energy systems is a challenge for today's industry. Systems are developed consisting of power hardware, control and communication hardware, and software of all kinds with more and more connectivity. Typically, engineering students at the Master's level specialize in technologies to apply later during their careers. However, a company's organisation and processes are far different from what students study in class. Product development implies several phases, from customer requirements definition to systematic verification and validation. Mondragon Unibertsitatea has identified the need to work with students on a product's whole life cycle, 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 could focus on different life cycle phases, such as requirements definition, architecture design and management, and continuous verification and testing. The curriculum has been integrated in a 4.5 ECTS course related to Systems Engineering in the Master's Degree in Smart Energy Systems at the Faculty Engineering of Mondragon Unibertsitatea.**

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

I. INTRODUCTION

Product development and manufacturing have several phases these days. Complex products consisting of many subsystems, which can be both hardware and software, are often commercialized in the energy field. In addition, the connectivity of Industry 4.0 makes today's systems cyber-physical. Computer-based algorithms control a set of mechanisms.

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

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

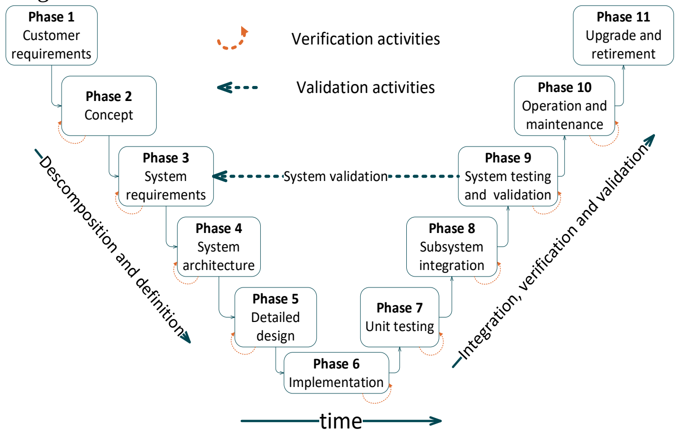


Figure 1 Electric scooter component architecture diagram

Until a few years ago, the cyber-physical systems development process was a document-centric methodology. The information generated in the process was collected and transmitted through documents.

This way of working has become increasingly difficult to manage because of today's complex 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. Companies have adopted a new way of working called Model-Based Systems Engineering (MBSE) in response to these difficulties. In the MBSE philosophy, a single virtual model represents the requirements, architecture, and system behaviour. MATLAB & Simulink are the main tools to model and simulate products under development in the energy systems field. Apart from being used for model-based design and simulation in recent years, this software incorporates systems engineering-related functionalities, such as requirements management (Requirements Toolbox™), architecture management (System Composer™), and systematic testing (Simulink Test™).

In this context, this article presents a curriculum development project that aims to bring MBSE into an engineering classroom. The aim of the curriculum is to provide teachers with a practical framework for teaching MBSE beyond classical theoretical content. This is achieved working in the MATLAB® & Simulink® environment. Specifically, it presents the design of different activities carried out for the Master of Science Degree in Smart Energy

Systems at Faculty of Engineering of Mondragon Unibertsitatea. [3]. The student is introduced to the application of the V-model processes using MathWorks tools through the master's curriculum in a 4.5 ECTS course related to Systems Engineering. The work focuses on the V-model's three phases: how to perform the requirements management (Phase 3) with the Requirements Editor, functional and formal decomposition (Phase 5) using System Composer, and validation against the requirements using Simulink Test (Phases 7-8). It should be noted that students on this master's course have already acquired knowledge of the design and manufacture of energy systems. Therefore, the detailed design and implementation phases (5 and 6) have not been covered on in this curriculum. Curriculum learning outcomes include:

- Managing the energy system's life cycle with the V-model and model-based techniques, and recognising the MBSE's ability to facilitate traceability.
- Writing and managing requirements with MATLAB® Requirements Editor.
- Composing system architectures and organising system components with System Composer.
- Arranging verification and validation tests against system requirements with test harnesses and test suites.

During the course, an electric scooter will be used as an example of a system. It is a system that the students have worked with before, so it is suitable for working on the requirements, architecture and validation phases. Figure 2 shows a block diagram of an electric scooter.

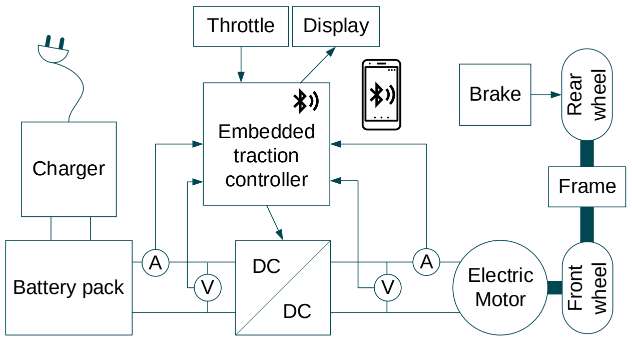


Figure 2 Electric scooter component architecture diagram

The structure of the article is as follows. Section II briefly introduces the V-model. Section III illustrates requirements basics and their management in Requirements Editor. Section IV presents an electric scooter's candidate architecture and the definition of different components in System Composer. Section V demonstrates the way to link behavioural and architecture models. Section VI explains using Test Harnesses and Test Manager to manage verification, validation, and testing following the V-model phases. Finally, section VII presents the conclusions. All the information and teaching material is available in a GitHub repository [4].

II. V-MODEL

As mentioned previously, the V-model is a standard methodology for life cycle management. The model is divided into two branches (Figure 1). The left branch contains conception processes, the requirements definition, and the design (high-level and detailed). This process leads to product development and manufacturing (hardware and software). In the right branch, integration, verification, and validation processes are performed. The time axis is bent to form a V in this model and each phase on the left branch is put at the same level as its counterpart in the right branch [5].

In the 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. 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 differences 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. The following question is answered during this process: Are we developing the product correctly? In short, it is about ensuring that the system that was built is well-designed, safe, and functions correctly. This process evaluates against internal requirements. Verification involves only one 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. The following question is answered during this process: 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.

III. SYSTEM REQUIREMENTS SPECIFICATION

It is time to write system requirements once customer requirements are received and a concept is proposed. In [6] 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).

Concerning 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, systems are becoming increasingly 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 [7], [8]

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.
Complete	It contains all the information to understand it.

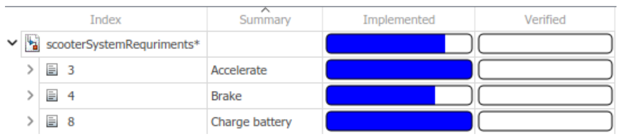
The IEEE Standards Style Manual [9] recommends using the term *shall* to indicate mandatory requirements to be followed strictly. The term *should* implies that among several possibilities, one is recommended without excluding others. Table 2 gives some examples of requirements writing.

Table 2 Requirement quality examples

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.	High

In MATLAB®, requirements are managed using the Requirements Editor from the Requirements Toolbox. This tool enables authoring and organising 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 requirements' main attributes are shown. In addition, the implementation and verification levels are also shown. Each requirement can be linked to a specific component once an architecture has been defined in Simulink (see Section III). Depending on the existing links, the editor sets a level of requirement implementation. In addition, the verification status is obtained from the test results programmed using Simulink Test (see Section VI).



Index	Summary	Implemented	Verified
scooterSystemRequriments*			
3	Accelerate		
4	Brake		
8	Charge battery		

Figure 3 System requirements defined in Requirements Editor

Each requirement has an attached sheet where all its information is gathered (see Figure 4). 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.

An interesting functionality 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.

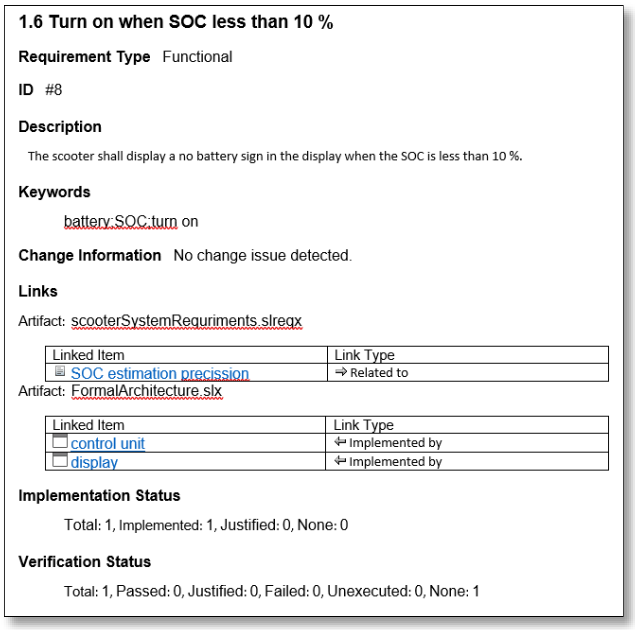
IV. SYSTEM ARCHITECTURE DESIGN

Phase 4 in the V-model life cycle is system architecture design. Once customer requirements are clear, a concept is proposed, and system requirements are 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 5 shows the electric scooter's formal architecture. It was designed following the system requirements presented in Section III. The formal architecture represents the system composition 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.

A. Architecture design

The basic element of creating an architecture is the component. Components are boxes that can be filled with more components or simulation behaviour models. As shown in Figure 5, components can have data inputs and outputs, but physical interfaces (diamond shaped) can also be used. With physical inputs, Simscape™ models can be integrated into the architecture.



1.6 Turn on when SOC less than 10 %

Requirement Type Functional

ID #8

Description

The scooter shall display a no battery sign in the display when the SOC is less than 10 %.

Keywords

battery;SOC;turn on

Change Information No change issue detected.

Links

Artifact: scooterSystemRequriments.slreqx

Linked Item	Link Type
SOC estimation precission	Related to

Artifact: FormalArchitecture.slx

Linked Item	Link Type
control unit	Implemented by
display	Implemented by

Implementation Status

Total: 1, Implemented: 1, Justified: 0, None: 0

Verification Status

Total: 1, Passed: 0, Justified: 0, Failed: 0, Unexecuted: 0, None: 1

Figure 4 Requirements sheet

B. Requirements Linking

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

C. Stereotypes

Stereotypes are used to define component or signal types so the architecture can be organized and analyzed more easily. For example, software and hardware components have been

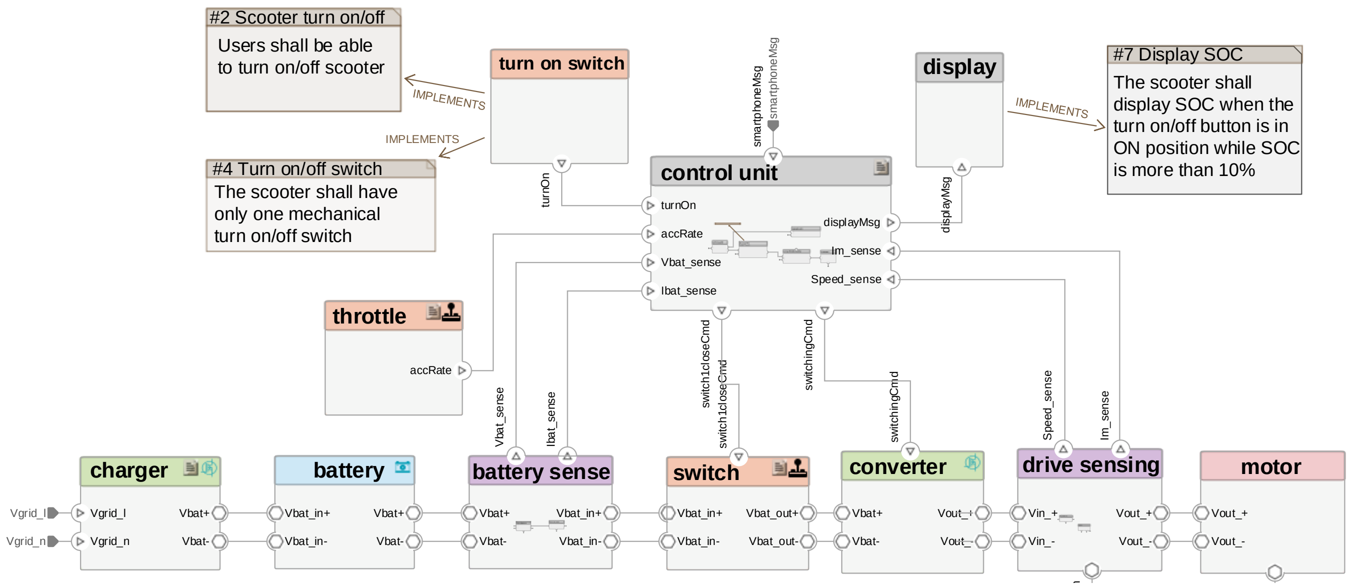
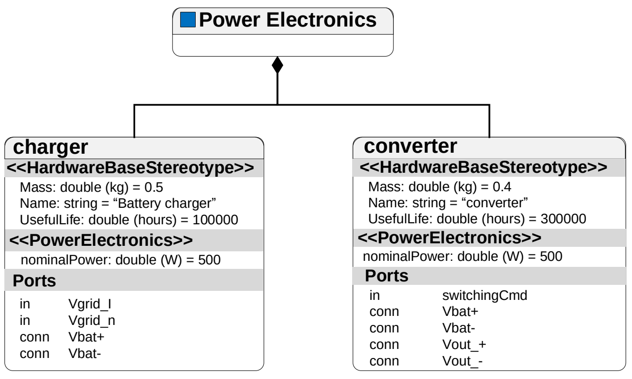


Figure 5 Scooter architecture in System Composer

defined in the scooter architecture. Within the software components, some may be dedicated to traction control, others dedicated to communications, and others to hardware input and output management. Components can be classified and facilitate development management by defining stereotypes. In the architecture shown in Figure 5, the components were classified into the following groups: storage systems, power electronics, sensing, electric machine, mechanical parts, actuators, and software. Moreover, each stereotype has its attributes such as mass, nominal power, or useful life. These attributes may be used to further analyze the proposed solution feasibility and compare different alternatives

D. *Views*

Once the architecture is defined and stereotypes applied, views help analyze 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.



charger
<<HardwareBaseStereotype>>
Mass: double (kg) = 0.5
Name: string = "Battery charger"
UsefulLife: double (hours) = 100000
<<PowerElectronics>>
nominalPower: double (W) = 500
Ports
in Vgrid_l
in Vgrid_n
conn Vbat+
conn Vbat-

converter
<<HardwareBaseStereotype>>
Mass: double (kg) = 0.4
Name: string = "converter"
UsefulLife: double (hours) = 300000
<<PowerElectronics>>
nominalPower: double (W) = 500
Ports
in switchingCmd
conn Vbat+
conn Vbat-
conn Vout_+
conn Vout_-

Figure 6 Embedded traction control strategy for electric scooter

Figure 6 shows a hierarchy view, where a filter was applied to see the power electronics components only. This tool is very useful for developers when they want to focus on a specific part of the design. Views are created interactively, but complete system information remains in the model. There is no need to create ad-hoc documents about the architecture for each team involved in the design with this model-based tool compared to a document-centric approach.

V. Linking Detailed Designs with Architectures

In principle, this curriculum's objective is not to work on the detailed design of embedded controls for electric drives. It is assumed that students have acquired sufficient knowledge to understand the hardware and software components of 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 7. It is composed by a cascade structure that controls the motor's speed and/or current/torque of the motor according to the reference established by the user in the throttle.

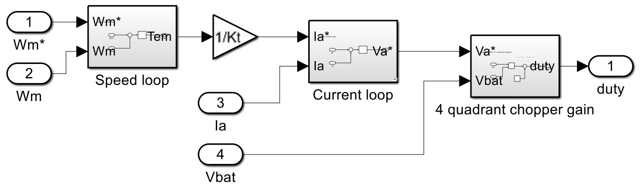


Figure 7 Embedded traction control strategy for electric scooter

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

Figure 8 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 presented in Figure 3, a column showing the implementation status also appears.

A full electric scooter model developed with System Composer and Simulink lets designers perform continuous verification and validation (V&V) activities during the whole life cycle, not only in the late phases of the V-model. Figure 9 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 MATLAB® Control System Toolbox™.

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.

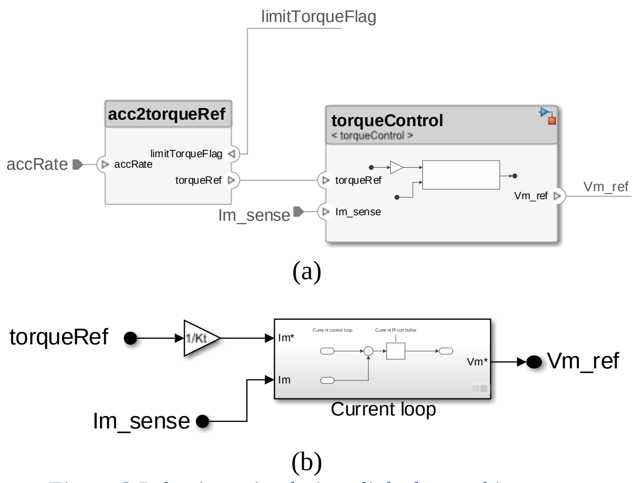


Figure 8 Behaviour simulations linked to architecture components (a) architecture component (b) behaviour model inside the torqueControl component

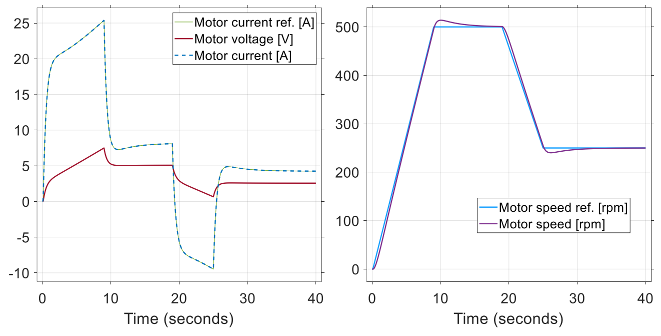


Figure 9 Current control simulation results

VI. 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 a Test Manager.

A. Test harnesses and unit tests

Unit testing isolates atomic components from the general design and tests their behaviour in standalone mode so 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 10 shows a test harness for the current control mentioned before.

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 instances are updated continuously. The harness shown in Figure 10 includes a test sequence and a test assessment block. These allow for implementing test inputs and verifying code programmatically with MATLAB® language. If the component under test needs any additional block to check its operation, it can be added here. The current loop uses a simplified motor transfer function to test its response in the example. In this way, there is no need to run all the components in the architecture, so unit testing is simplified.

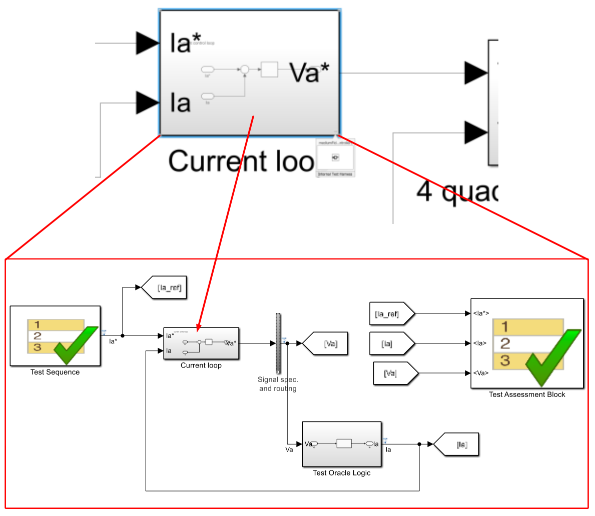


Figure 10 Current control test harness

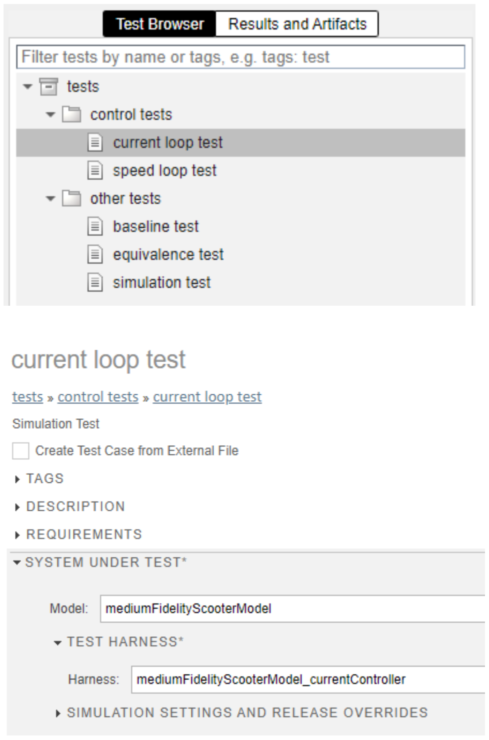
Apart from the test sequence and assessment blocks, several alternatives can be used to configure inputs and check outputs, such as Signal Builders or charts or signals imported from the workspace. Finally, it is worth mentioning that test harnesses can also be stored in a separate file, making it easier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

B. 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, this can lead to many tests and results in complex systems. The example used in this curriculum project in a first approximation of system requirements (top-level requirements) already has 43. If 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. Tests are grouped by test files, suites, and cases (Figure 11). 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.

Once tests are configured, the Test Manager executes them in batch. Simulations run in the background, and results are 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 test case results. Thanks to this feature, students have a powerful resource to understand how test cases are linked to requirements, how Model-Based tools can help validate designs, and how traceability tools can help manage complex system development.


Test Browser | Results and Artifacts

Filter tests by name or tags, e.g. tags: test

- tests
  - control tests
    - current loop test
    - speed loop test
  - other tests
    - baseline test
    - equivalence test
    - simulation test

current loop test

tests » control tests » current loop test

Simulation Test

Create Test Case from External File

TAGS

DESCRIPTION

REQUIREMENTS

SYSTEM UNDER TEST*

Model: mediumFidelityScooterModel

TEST HARNESS*

Harness: mediumFidelityScooterModel_currentController

SIMULATION SETTINGS AND RELEASE OVERRIDES

Figure 11 Test Manager and test case configuration

As an example, Figure 12 shows the results of a test where current's overshoot is checked. A test was programmed where Simulink Test automatically evaluates if the current's response has an overshoot greater than 20 %.

This is the result of simulating the test harness shown in Figure 10. 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).

VII. CONCLUSIONS

This article shows curriculum development for working with MBSE in engineering studies. The necessary resources have been created to work on the most important aspects of this field, Based on the V-model for life cycle management and MATLAB & Simulink. A student's primary learning outcome was learning to manage the energy systems' life cycle using the V-model and model-based techniques.

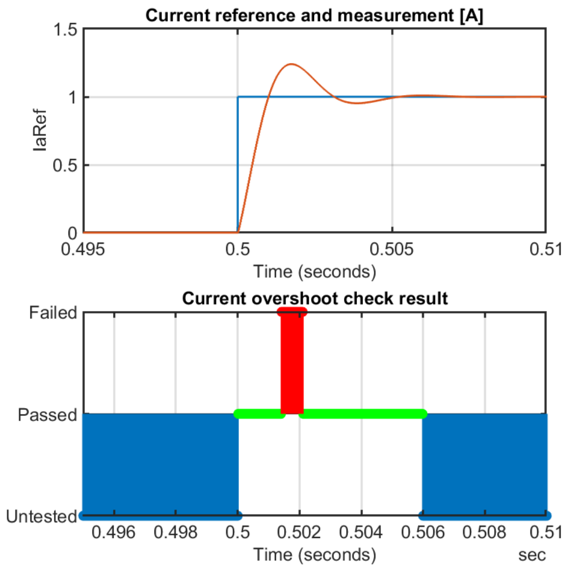


Figure 12 Test Manager and test results

First, creating a set of requirements in the Requirements Editor was shown following the guidelines established in the IEEE Standards Style Manual. Next, 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.

In conclusion, the tools integrated between MATLAB® & Simulink® can be a suitable starting point for working on MBSE in the classroom. Academic institutions usually use this software in the classroom in the energy and electronics field. The curriculum and related activities have been integrated into a subject linked to Systems Engineering during the 2022-2023 academic year. This first experience has shown that if students are familiar with this software, they can begin to develop competences linked to systems engineering without the need for specific software training.

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] Mondragon Unibertsitatea, "Smart Energy Systems Master's Degree." https://www.mondragon.edu/en/master-degree-smart-energy-systems

[4] "GitHub - Model-based life cycle with MATLAB & Simulink." https://github.com/MU-MATHWORKS/MODEL-BASED-LIFE-CYCLE-WITH-MATLAB-SIMULINK

[5] T. Weilkiens, J. G. Lamm, S. Roth, and M. Walker, *Model-Based System Architecture*. in Wiley Series in Systems Engineering and Management. Wiley, 2015. [Online]. Available: https://books.google.es/books?id=w1TKCQAAQBAJ

[6] IEEE Standards Association, “IEEE Standard for Application and Management of the Systems Engineering Process, IEEE 1220-2005,” 2005.

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

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

[9] 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