Mondragon
Unibertsitatea

Faculty of
Engineering

FUNDAMENTALS OF MODEL-BASED SYSTEMS ENGINEERING FOR LIFE CYCLE MANAGEMENT

Testing and validation platforms

MsC in Smart Energy Systems

TABLE OF CONTENTS

1 ABBREVIATIONS 2

2 V-MODEL FOR LIFE CYCLE MANAGEMENT 2

3 MODEL-BASED SYSTEMS ENGINEERING 6

3.1 MODELS AS THE CENTER FOR PRODUCT DEVELOPMENT 6
3.2 TOOLS FOR MODEL-BASED DEVELOPMENT 11
3.2.1 MODEL-IN-THE-LOOP 11
3.2.2 SOFTWARE-IN-THE-LOOP 12
3.2.3 PROCESSOR-IN-THE-LOOP 12
3.2.4 HARDWARE-IN-THE-LOOP 12
3.2.5 RAPID CONTROL PROTOTYPING (RCP) 13
3.3 MATLAB & SIMULINK FOR V-MODEL APPLICATION 13

4 BIBLIOGRAPHICAL REFERENCES 17

1 Abbreviations

AC	Alternating current
DC	Direct current
HIL	Hardware-in-the-loop
MBD	Model-Based design
MBSE	Model-Based systems engineering
MIL	Model-in-the-loop
PIL	Processor-in-the-loop
SIL	Software-in-the-loop
RCP	Rapid Control Prototyping

2 V-model for life cycle management

The development and manufacture of a product can have several phases. In the field of smart 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.

As an example of such a system, Figure 1 shows a block diagram of an electric scooter. Even if it does not seem such a complex system, the architecture hierarchy in Figure 2 already shows some complexity.

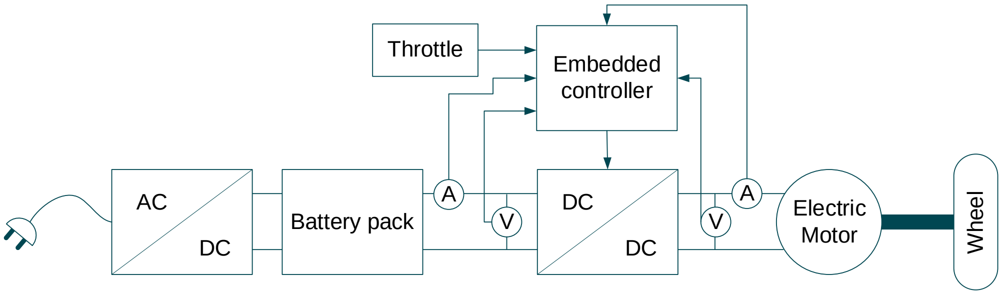


Figure 1 Block diagram of electric scooter

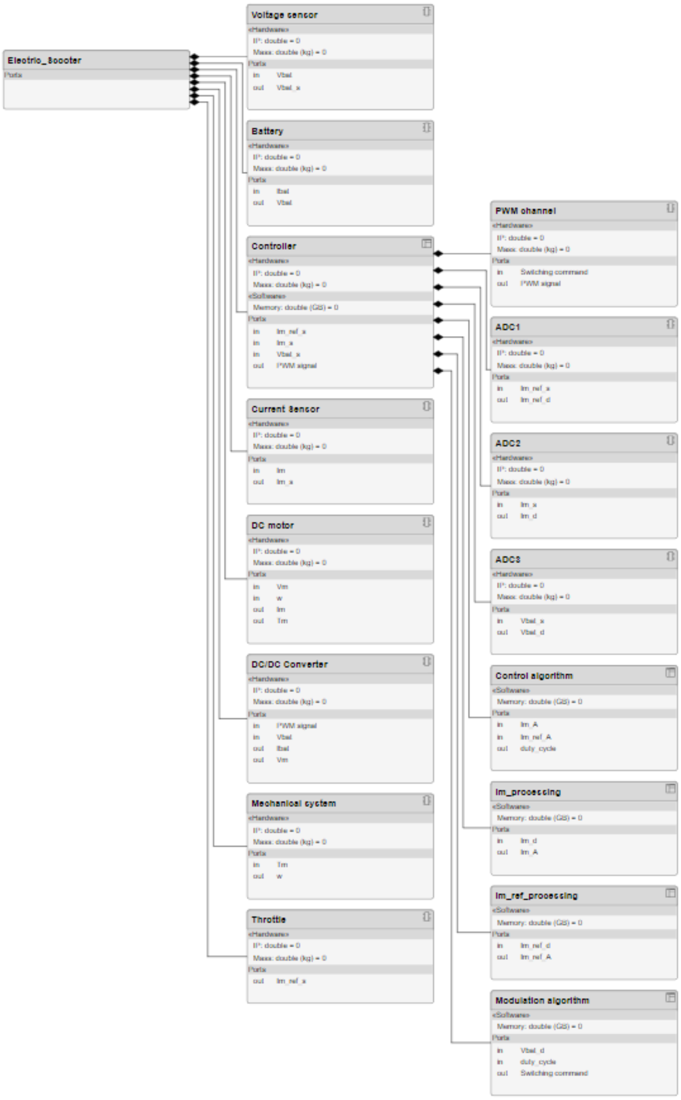


Figure 2 Architecture hierarchy of the electric scooter

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. 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. Appendix **¡Error! No se encuentra el origen de la referencia.** presents other life cycle management methodologies.

This model (Figure 3) describes the life cycle from product conception to operation and maintenance. 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.

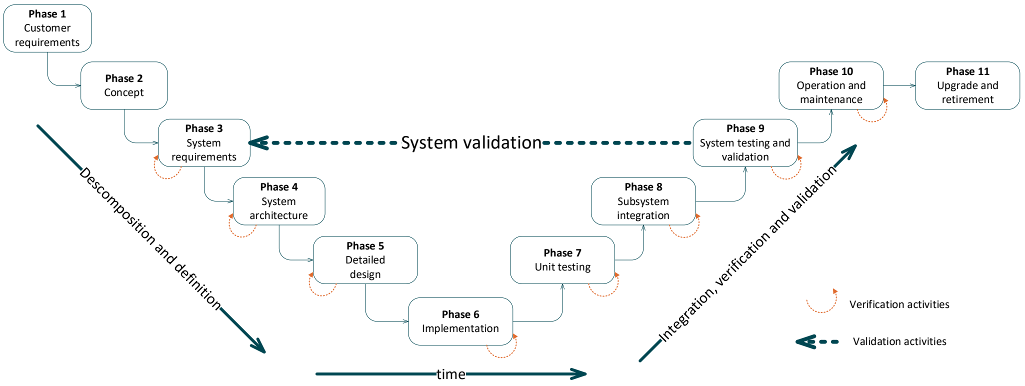


Figure 3 V-model diagram

Each phase of this model is described below:

- 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 necessary to be clear about the difference between verification and validation activities.

On the one 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.

On the other hand, verification is 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.

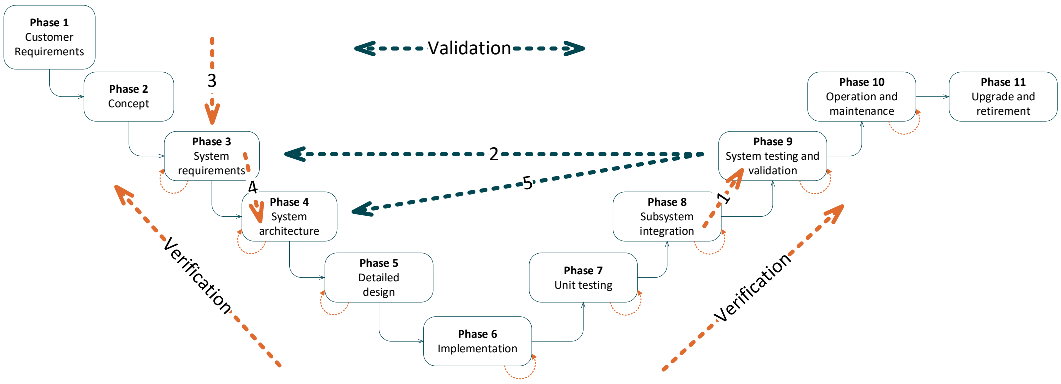
When we are faced with several activities and don’t know if they are verification or validation tasks, we should assess if they involve two consecutive phases (from phase n to phase n+1) or they involve parallel phases in the life cycle. A verification task tries to check if we are ready to move on to the next phase. Hence, this is represented in Figure 3 with arrows closing loops in the same phase. However, a validation task checks if the system works as requirements said, so may involve phase 3 and phase 9 in Figure 3.

Finally, Table 1 shows a classification of different tasks in the development of an electric scooter.

Table 1 Examples of verification and validation activities

id	Tasks	Verification	Validation	Notes
1	Inspection check to see if electric scooter is ready to test driving	X		Subsystems were integrated and test drive is part of phase 9 system testing.
2	Compile test results and write documentation showing fulfilment of safety requirements		X	Involves phases 3 and 9 of the V-model.
3	Participation in workshop to define requirements and standards	X		Part of phase 3
4	Control that all safety requirements are included in the design documents	X		Involves checking if phase 4 takes into account results of phase 3.
5	Detailed inspection of quality according to design documents		X	Checks in phase 9 respect to documents in phase 4.

InvFigure 4 shows the location of each phase in the V-model diagram.



InvFigure 4 Validation and verification activities mapping in V-model

3 Model-based systems engineering

3.1 Models as the center for product development

The V-model described above, until a few years ago, 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. Figure 5 shows a simplified representation of the viewpoints depending on the professional looking at them.

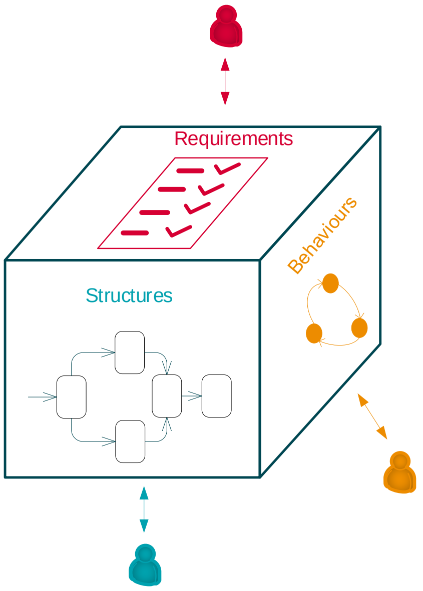


Figure 5 Viewpoints of a system (Modified from [3])

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 model represents the requirements, architecture and behavior of a system. INCOSE [4] describes MBSE as “the formalized application of modelling to support system requirements, design, analysis, verification and validation activities beginning in the conceptual design phase and continuing throughout development and later life cycle phases”.

According to [5] a model is a physical, mathematical or logical representation of a system, entity, phenomenon or process. In the previous section, the word model referred to a way of representing a process, specifically, the life cycle. However, from now on, by model we will refer to the representation of a system, in our case, en energy systems. In this field, a model is a virtual replica of a energy system capable of representing its structure and dynamic behaviour.

Models have been used in the design and development phases for years. Modelling is an essential step for the development of feedback controls for dynamic systems. In addition, development and simulation software such as MATLAB & Simulink allows designing, analyzing and simulating dynamic models in a simple and fast way.

The advantage of using MBSE is obvious because a model shared by the whole development team centralizes all the viewpoints shown in Figure 5. Moreover, the communication of information is based on visual modelling, easier to understand [6]. This approach allows working without having to generate additional documentation. Someone who is working on requirements tracking may only be interested in the validation results, while someone who is working on the design of a particular component will want to see the specification of the interfaces or its dynamic behaviour.

Different techniques have been proposed to model complex systems and make MBSE reality. One of them is SysML (System Modelling Language), a variant of UML (Unified Modelling Language) used in software engineering. SysML establishes standardized diagrams to represent systems from different points of view (see Figure 6).

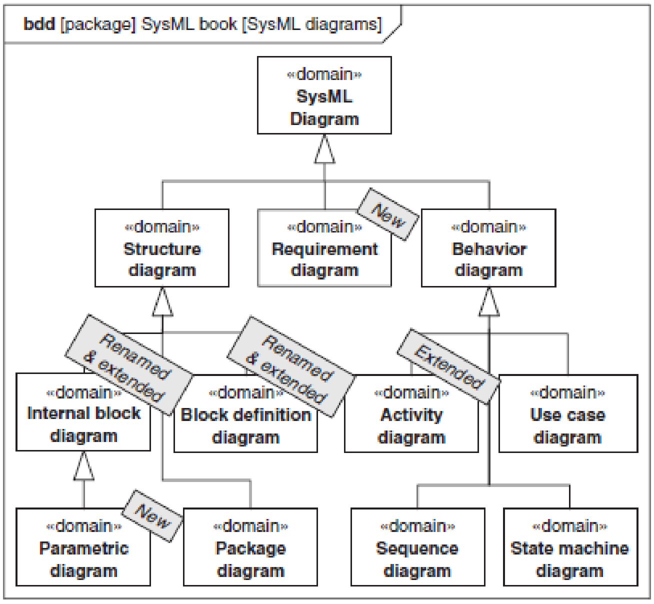

bdd [package] SysML book [SysML diagrams]
«domain» SysML Diagram
«domain» Structure diagram
«domain» Requirement diagram
New
«domain» Behavior diagram
Renamed & extended
Renamed & extended
Extended
«domain» Internal block diagram
«domain» Block definition diagram
«domain» Activity diagram
«domain» Use case diagram
New
«domain» Parametric diagram
«domain» Package diagram
«domain» Sequence diagram
«domain» State machine diagram

Figure 6 SysML diagrams (and differences respect to UML) [7]

As stated before, the system is viewed from three perspectives. The requirement diagram (req) in Figure 7 represents the requirements view.

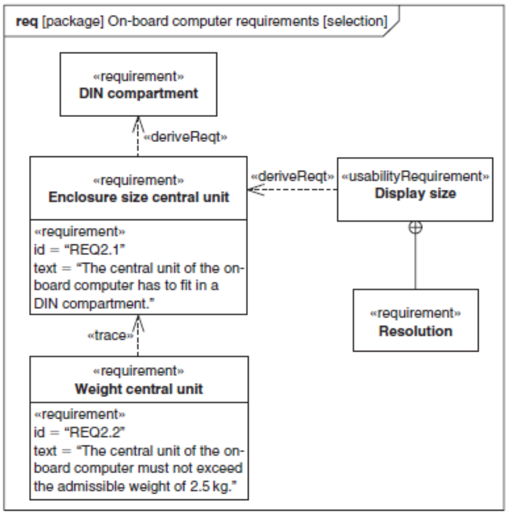
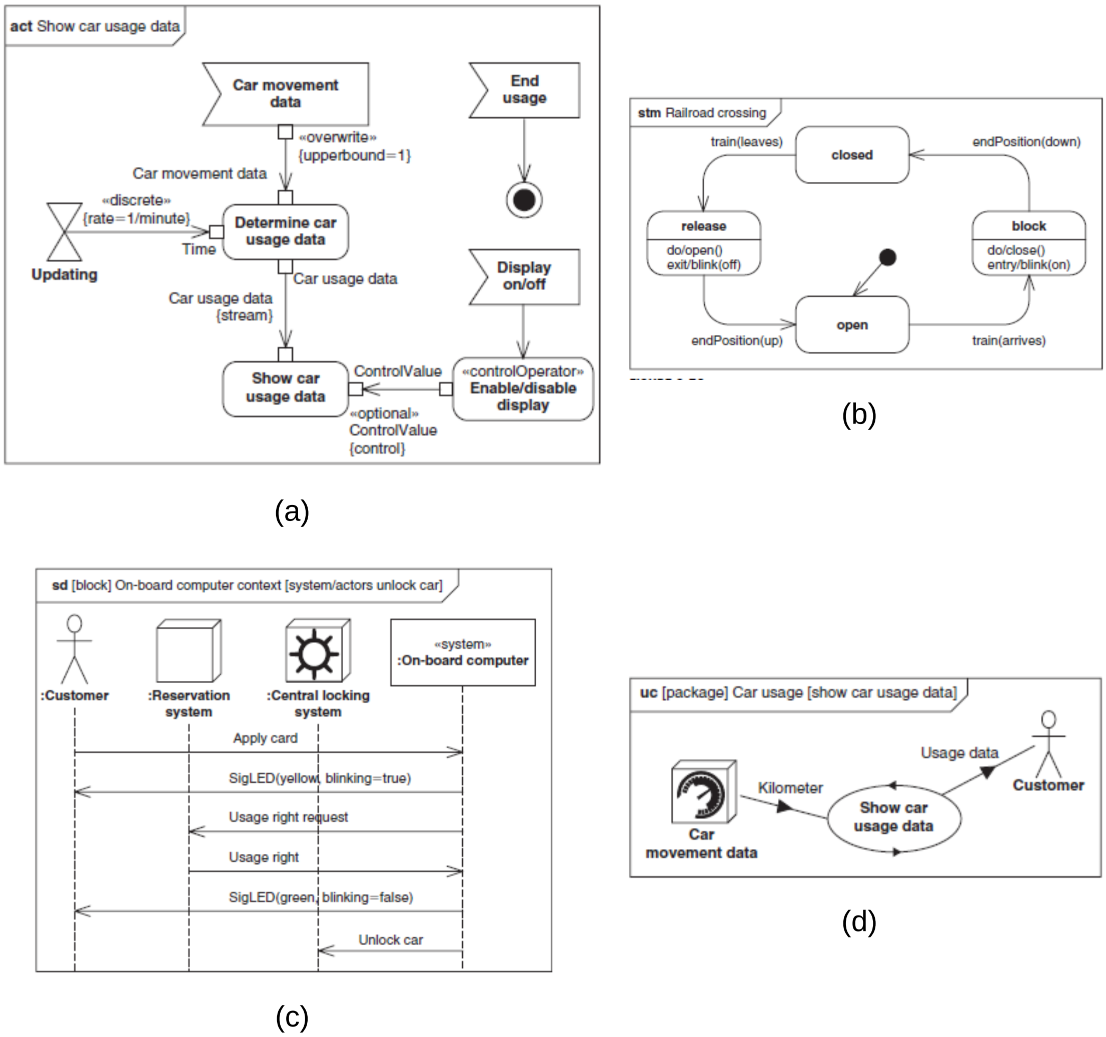

req [package] On-board computer requirements [selection]
«requirement»
DIN compartment
«deriveReqt»
«requirement»
Enclosure size central unit
«requirement»
id = "REQ2.1"
text = "The central unit of the on-board computer has to fit in a DIN compartment."
«deriveReqt»
«usabilityRequirement»
Display size
«requirement»
Resolution
«trace»
«requirement»
Weight central unit
«requirement»
id = "REQ2.2"
text = "The central unit of the on-board computer must not exceed the admissible weight of 2.5 kg."

Figure 7 Requirements diagram [7]

Activity (act), sequence (sd), use case (uc) and state machine (st) diagrams can represent behaviour.



(a)

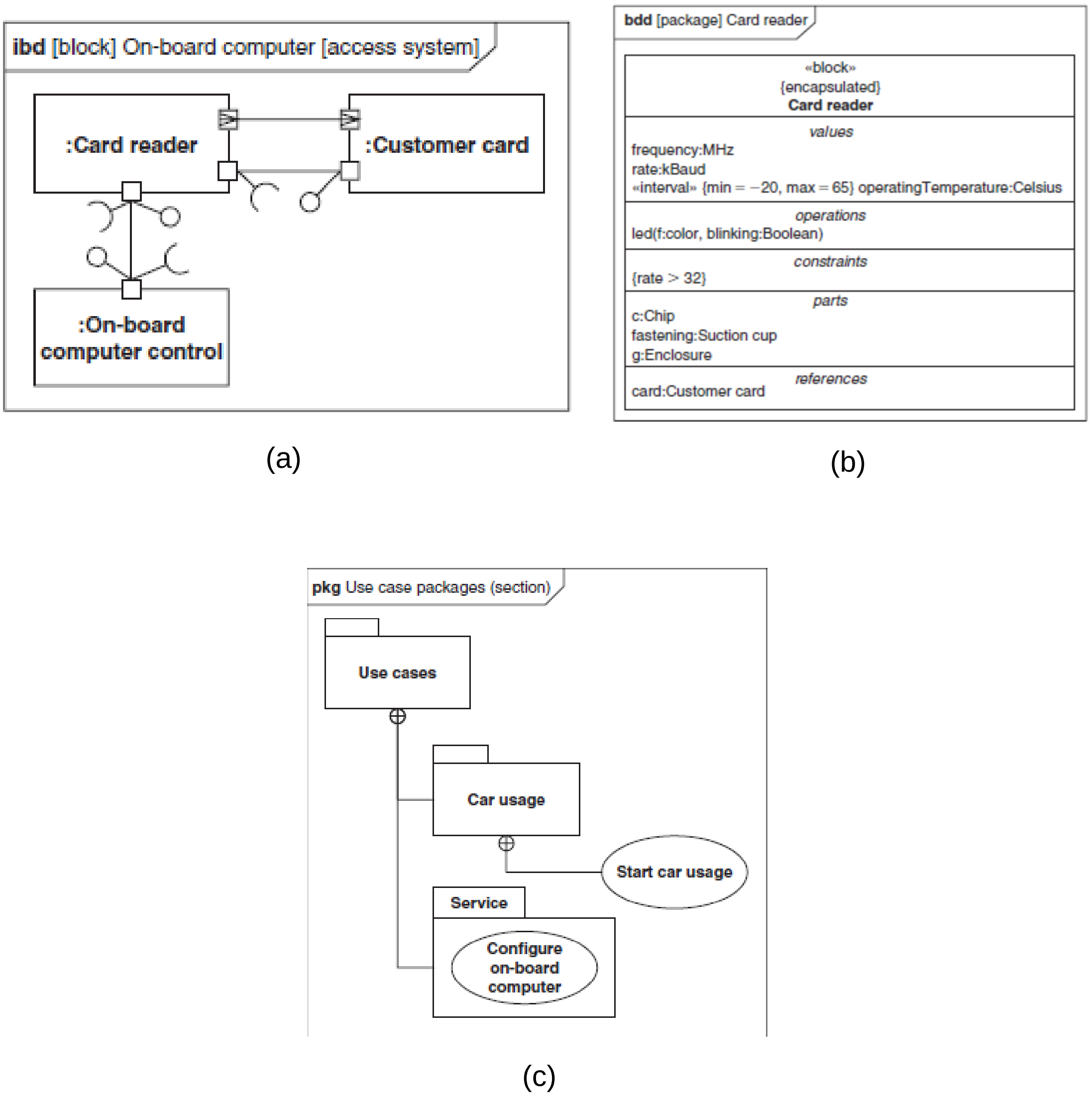
(b)

(c)

(d)

Figure 8 Behavioural SysML diagrams. (a) Activity diagram (b) State machine diagram (c) Sequence diagram (d) Use Case diagram [7]

Finally, structure is represented by internal block (ibd), package (pkg) and block definition (bdd) diagrams.


ibd [block] On-board computer [access system]
:Card reader
:Customer card
:On-board computer control
bdd [package] Card reader
«block»
{encapsulated}
Card reader
values
frequency:MHz
rate:kBaud
«interval» {min = −20, max = 65} operatingTemperature:Celsius
operations
led(f:color, blinking:Boolean)
constraints
{rate > 32}
parts
c:Chip
fastening:Suction cup
g:Enclosure
references
card:Customer card
pkg Use case packages (section)
Use cases
Car usage
Start car usage
Service
Configure on-board computer

(a)

(b)

(c)

Figure 9 Structural SysML diagrams (a) Internal block diagram (b) Block definition diagram (c) Package diagram [7]

The main advantages of MBSE are [8]:

- Improvement of communication in multidisciplinary teams. Today in the development of cyber-physical systems, professionals from many disciplines are involved: electrical engineers, mechanical engineers, software engineers, system architects... They all use different languages and tools so having a unified model of the system helps to coordinate everyone's work, avoid confusion and ambiguities.
- It allows companies to design and simulate their products before manufacturing them or having prototypes. This advantage has been exploited for decades in phases 4 and 5 of the life cycle, where simulation tools are essential to analyse the performance of proposed architectures and topologies at a reduced cost. Thus, MBSE makes it possible to analyse and advance decisions that would previously have been taken on the basis of prototypes in phase 6.

- It facilitates the management of system complexity. Thanks to the models, a system can be viewed from different points of view quickly and easily. There is a unique master model that everyone can look at from its own point of view.
- Documents can be generated automatically from the model, making their maintenance easier and reducing costs and effort.

3.2 Tools for model-based development

With the advancement of MBSE, different tools for system validation and verification have emerged. In addition to representing the system using standards such as SysML and simulating it with software such as MATLAB & Simulink, simulation platforms have been extended to the entire life cycle. Models are used both to replace the system to be controlled and to have an initial version of the control before its coded.

There are different simulation techniques for model-based verification, such as Model-in-the-loop (MIL), Software-in-the-loop (SIL), Processor-in-the-loop (PIL), Hardware-in-the-loop (HIL) and Rapid Control Prototyping (RCP). The characteristics and structure of each of these strategies are detailed below for a classical control application as shown in the Figure 10.

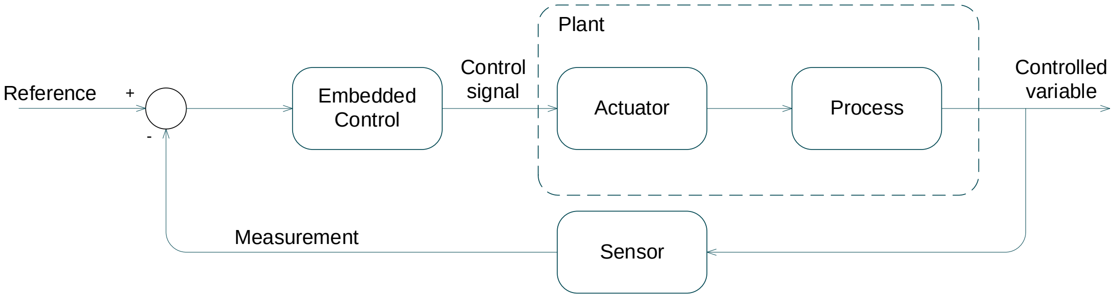


Figure 10 Basic control system block diagram

3.2.1 Model-in-the-loop

In the Model-in-the-Loop technique the complete system, both the control and the plant, is simulated.

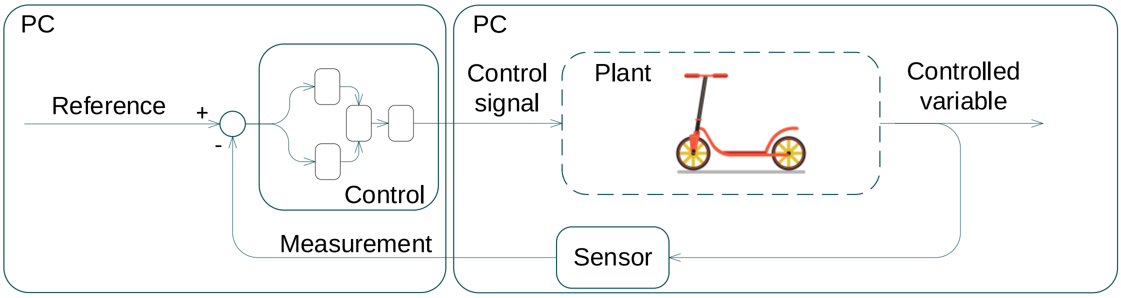


Figure 11 MIL structure

For the MATLAB & Simulink case, it would be a matter of having a plant model and a control model. The control model refers to a strategy implemented by block diagrams or state machines. In this scenario, the control is not yet coded. The usual is a pure computer simulation, but there may be cases where the simulations are done in real-time simulators.

3.2.2 Software-in-the-loop

The Software-in-the-loop technique consists of implementing the control in code. The product is still implemented on text platforms, such as a computer or a real-time simulator, but the implementation is real. The code is close to the final version that will be shipped on the control hardware. This scenario allows to verify the code, the execution sequences and even to size the memory needed in the future processor in real time.

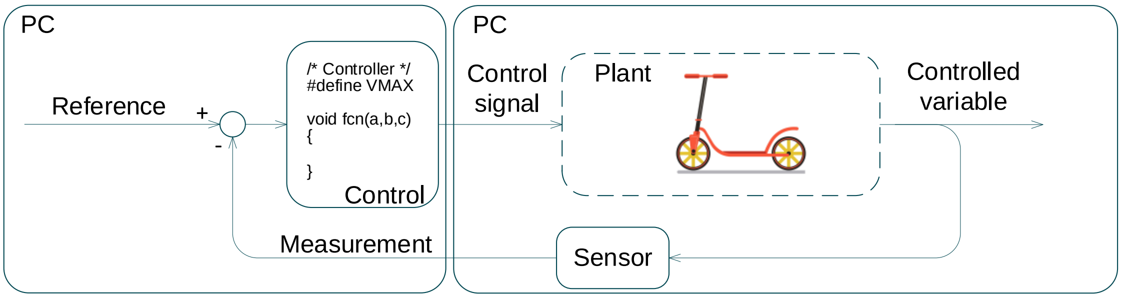


Figure 12 SIL structure

3.2.3 Processor-in-the-loop

In this scenario, the control is already implemented in a hardware controller, although as a prototype. In this way, run times, memory usage etc. can be measured. In control applications, since the controller will have to read or write analogue or digital signals, it may be necessary for the plant to be simulated in a real-time simulator. This requires the plant to run and give a response below the minimum execution time of the control.

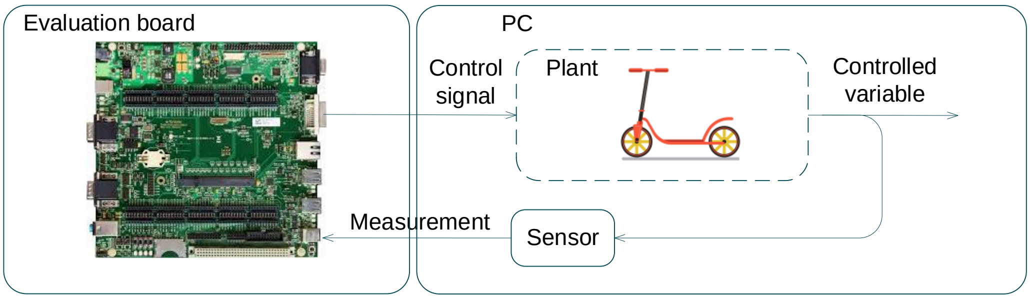


Figure 13 PIL structure

3.2.4 Hardware-in-the-loop

This tool allows to validate the hardware and control software without the need of having the real plant. The simulator has the analogue and digital inputs and outputs to communicate with the controller, so that it can run the software in the same way as in the real application. Depending on the controller, the simulator will run the model of the plant in real time, even communications of all kinds can be implemented to emulate the real controlled system

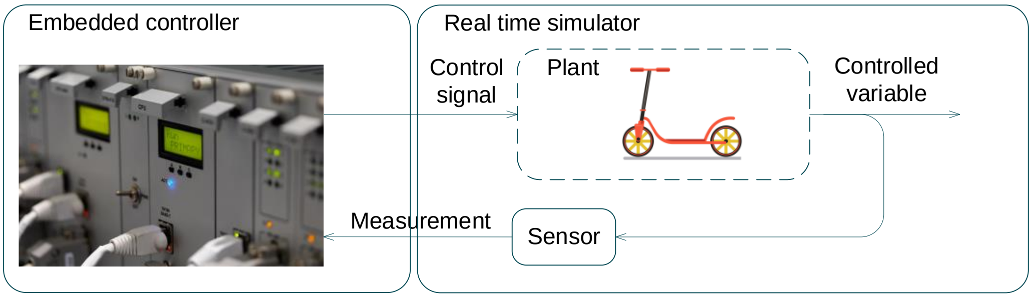


Figure 14 HIL structure

3.2.5 Rapid control prototyping (RCP)

In cases where the system to be controlled already exists, this technique allows experimental implementations of the control strategy in test environments. Without the need to have the final control hardware, the control strategy can be tested on laboratory equipment to speed up the verification process.

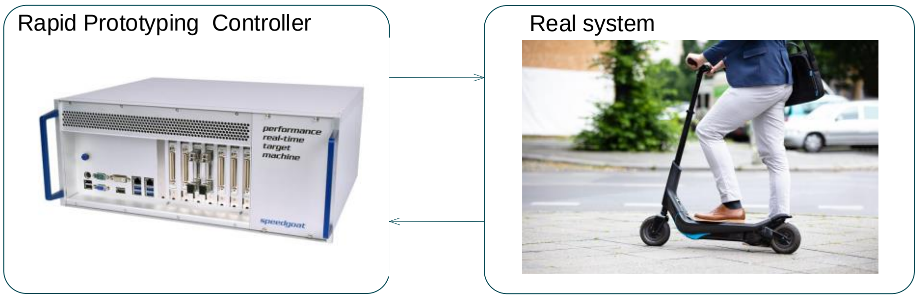

Rapid Prototyping Controller
performance real-time target machine
speedgoat
Real system

Figure 15 RCP structure

3.3 MATLAB & Simulink for V-model application

Traditionally, in the field of energy systems and the control of dynamic systems, MATLAB & Simulink has been the software of reference for their development. Until a few years ago, this software was used to analyze the behavior of systems by representing and simulating them with block diagrams. But lately, there are more and more add-ons and toolboxes that allow to integrate all the views shown in Figure 6 in a single simulation project. It can be said that dynamic simulation software and MBSE are finding each other and joining forces to improve and facilitate the development of complex systems.

Figure 16, Figure 17 and Figure 18 show block diagrams built with Simulink System Composer as an alternative to SysML diagrams. The big advantage of this kind of tools is that they integrate the usual behavioural (dynamic) model with architectural and requirements information.

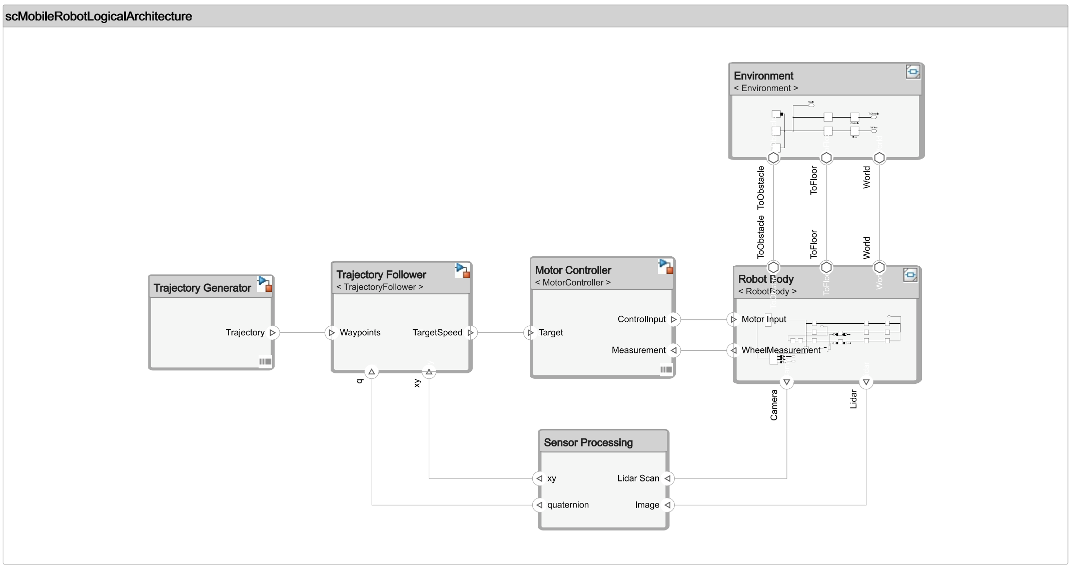


Figure 16 Block diagram and dynamic simulation model designed with System Composer [9]

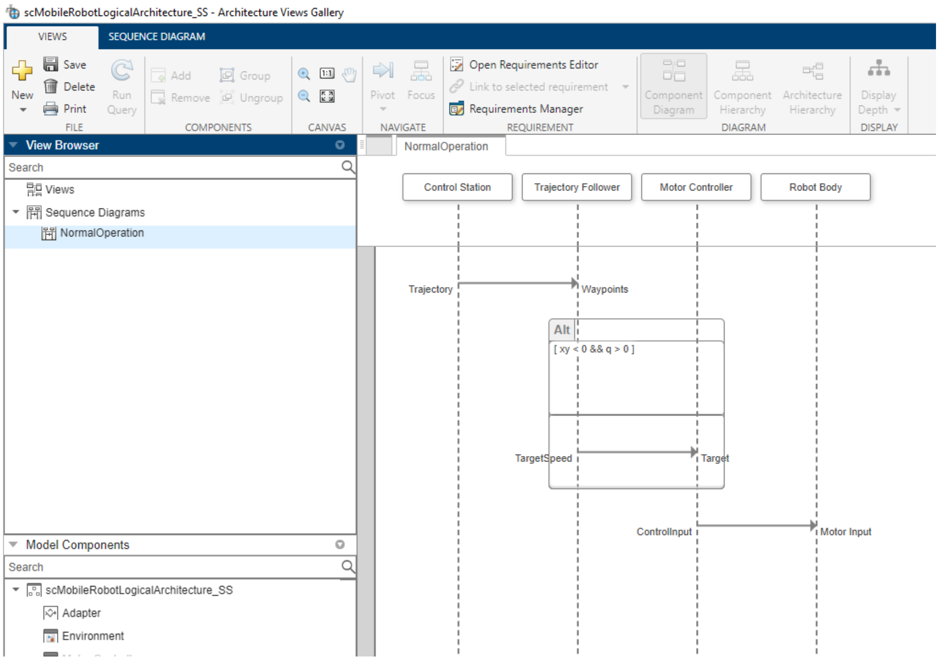
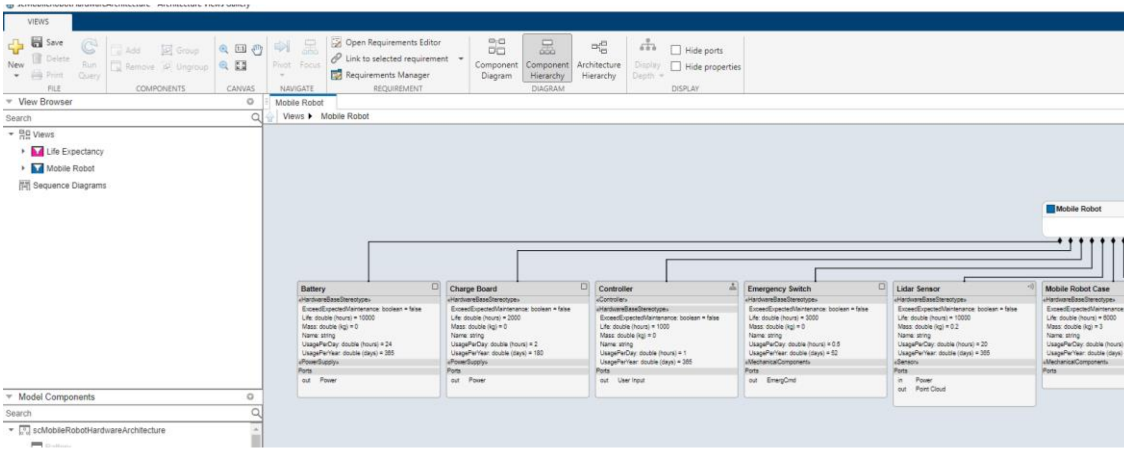


Figure 17 Sequence diagram generated with System Composer [9]



Battery
«HardwareBaseStereotype»
ExceedExpectedMaintenance: boolean = false
Life: double (hours) = 10000
Mass: double (kg) = 0
Name: string
UsagePerDay: double (hours) = 24
UsagePerYear: double (days) = 365
«PowerSupply»
Ports
out Power

Charge Board
«HardwareBaseStereotype»
ExceedExpectedMaintenance: boolean = false
Life: double (hours) = 2000
Mass: double (kg) = 0
Name: string
UsagePerDay: double (hours) = 2
UsagePerYear: double (days) = 180
«PowerSupply»
Ports
out Power

Controller
«Controller»
«HardwareBaseStereotype»
ExceedExpectedMaintenance: boolean = false
Life: double (hours) = 1000
Mass: double (kg) = 0
Name: string
UsagePerDay: double (hours) = 1
UsagePerYear: double (days) = 365
Ports
out User Input

Emergency Switch
«HardwareBaseStereotype»
ExceedExpectedMaintenance: boolean = false
Life: double (hours) = 3000
Mass: double (kg) = 0
Name: string
UsagePerDay: double (hours) = 0.5
UsagePerYear: double (days) = 52
«MechanicalComponent»
Ports
out EmergCmd

Lidar Sensor
«HardwareBaseStereotype»
ExceedExpectedMaintenance: boolean = false
Life: double (hours) = 10000
Mass: double (kg) = 0.2
Name: string
UsagePerDay: double (hours) = 20
UsagePerYear: double (days) = 365
«Sensor»
Ports
in Power
out Point Cloud

Mobile Robot Case
«HardwareBaseStereotype»
ExceedExpectedMaintenance
Life: double (hours) = 6000
Mass: double (kg) = 3
Name: string
UsagePerDay: double (hours)
UsagePerYear: double (days)
«MechanicalComponent»
Ports

Figure 18 Hierarchy diagram generated with System Composer [9]

MathWorks provides multiple toolboxes that can be used in different phases of the V-model process. With the use of MATLAB & Simulink and its toolboxes, improvements in the final product can be achieved, due to the advantages of this type of design.

Some toolboxes that can be identified as useful for V-model application are these:

- **Requirements Toolbox:** this allows you to create, link and validate requirements in MATLAB or Simulink. You can create requirements using text with custom attributes, or import them from requirements management tools.
- **System composer:** this toolbox allows you to specify and analyse architectures for model-based system engineering and software architecture modelling. With System Composer, you can map requirements while refining an architectural model that can then be designed and simulated in Simulink.
- **Stateflow:** it provides a graphical language that includes state transition diagrams, flowcharts, state transition tables, and truth tables. Stateflow can be used to describe how MATLAB algorithms and Simulink models react to input signals, events, and time-based conditions.
- **Embedded coder:** generates readable, compact, and fast C and C++ code for embedded processors used in mass production. It extends MATLAB & Simulink Coder with advanced optimizations for precise control of functions, files and generated data. These optimizations increase code efficiency and facilitate integration with legacy code, data types and calibration parameters. You can incorporate a third-party development tool and create an executable for turnkey deployment on the embedded system or rapid prototyping board.
- **Simulink Test:** provides tools for creating, managing and executing systematic tests based on simulations of models, generated code and simulated or physical hardware. It includes simulation, benchmark and equivalence test templates that enable functional, unit, regression and consequential testing using software-in-the-loop (SIL), processor-in-the-loop (PIL) and hardware-in-the-loop (HIL) modes in real time.
- **Simulink Report Generator:** it allows you to create web views with which you can view, navigate and share Simulink models from a web browser and without a Simulink license. You can integrate web views into HTML code generation, requirements, coverage and other reports.

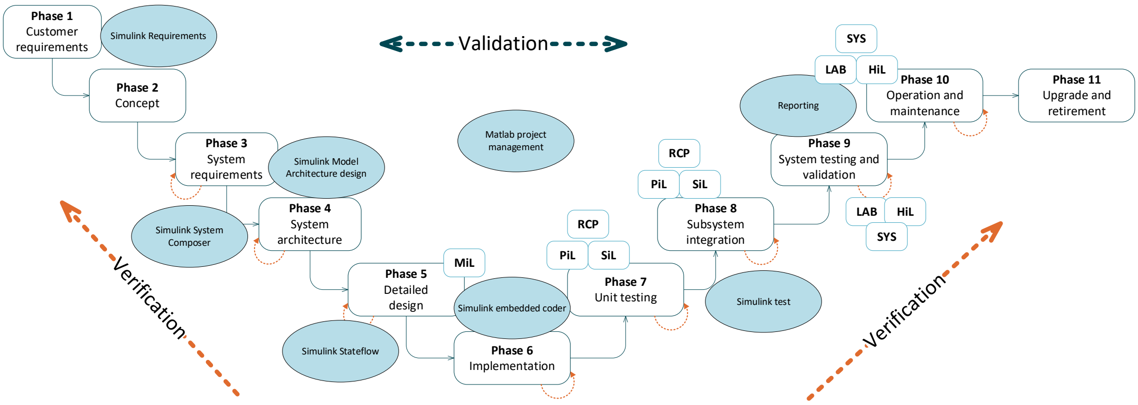


Figure 19 MATLAB & Simulink applications for model-based development and V-model

Taking as an example the development of a control unit for electric drives (software + hardware), the tools described above are usually used as follows:

- In the design phases there is no control software or hardware. The MIL technique allows the architecture and design of each subsystem to be simulated and analysed in detail.
- In the unit testing phase, hardware and software continue to be tested separately. On the software side, there are several options. With SIL, the control code is integrated into the simulation or with HIL, the code can be tested on a test control board against a simulated plant in real time. Using RCP, the control code can be tested against the real (or laboratory) plant on a test controller.
- In the subsystem verification phase, the same tools as in the previous phase are used, but components as a whole or complete subsystems can be tested.
- In the integration phase, the control software and hardware are gradually integrated and tested in HIL, laboratory and real scenario.
- In the acceptance and validation phase, the same tools are used to ensure compliance with system requirements.
- In operation and maintenance, the previously mentioned tools continue to be used to resolve incidents or design improvements.

4 Bibliographical 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] “Model Based Systems Engineering (MBSE) on AWS: From Migration to Innovation.”

[4] INCOSE, “SYSTEMS ENGINEERING VISION 2020,” 2007. Accessed: Jan. 04, 2023. [Online]. Available: https://sdincose.org/wp-content/uploads/2011/12/SEVision2020_20071003_v2_03.pdf

[5] J.S. Gansler, “DoD Modeling and Simulation (M&S) Glossary,” 1998.

[6] Chantal Cappelletti, Simone Battistini, and Benjamin K. Malphrus, Eds., *Cubesat Handbook*. Elsevier, 2021. doi: 10.1016/C2018-0-02366-X.

[7] Tim Weilkiens, *Systems Engineering with SysML/UML*. Elsevier, 2007. doi: 10.1016/B978-0-12-374274-2.X0001-6.

[8] J. Holt, S. A. Perry, and M. Brownsword, *Model-Based Requirements Engineering*. Institution of Engineering and Technology, 2011. doi: 10.1049/PBPC009E.

[9] MathWorks, “Simulate Mobile Robot with System Composer Workflow.” https://www.mathworks.com/help/systemcomposer/ug/mobile-robot-workflow.html (accessed Jan. 04, 2023).

[10] W. W. Royce, “Managing the Development of Large Software Systems,” in *ICSE ’87: Proceedings of the 9th international conference on Software Engineering*, 1987, pp. 328–338. Accessed: Jan. 04, 2023. [Online]. Available: https://dl.acm.org/doi/10.5555/41765.41801

[11] N. D. Birrell and M. A. Ould, *A Practical Handbook for Software Development*. Cambridge University Press, 1985. doi: 10.1017/CBO9780511624223.