FUNDAMENTALS OF MODEL-BASED DESING FOR LIFE CYCLE MANAGEMENT

Testing and validation platforms

MsC in Smart Energy Systems

**ÍNDICE**

[1 Glossary 2](#_Toc108519748)

[2 V-model for Life Cycle Management 2](#_Toc108519749)

[2.1 Fundamentals 2](#_Toc108519750)

[2.2 Use case: Electric scooter development using the V-model Life Cycle 5](#_Toc108519751)

[3 Desarrollo basado en modelos 5](#_Toc108519752)

[3.1 Herramientas para el desarrollo basado en modelos 5](#_Toc108519753)

[3.1.1 Model-in-the-loop 6](#_Toc108519754)

[3.1.2 Software-in-the-loop 6](#_Toc108519755)

[3.1.3 Processor-in-the-loop 7](#_Toc108519756)

[3.1.4 Hardware-in-the-loop 7](#_Toc108519757)

[3.1.5 Prototipado rápido de control 8](#_Toc108519758)

[3.2 Desarrollo basado en modelos y modelo en V 8](#_Toc108519759)

[4 MATLAB & Simulink for V-model application 10](#_Toc108519760)

[5 Anexo: Modelos de gestión del Ciclo de Vida 10](#_Toc108519761)

[5.1.1 Waterfall 10](#_Toc108519762)

[5.1.2 B-model 12](#_Toc108519763)

[5.1.3 Spiral model 14](#_Toc108519764)

[6 Referencias bibliográficas 15](#_Toc108519765)

# Glossary

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

# 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 the functional decomposition of a electric car.

|  |
| --- |
| ISO 26262 |
| Figure 1 Functional decomposition of a electric car |

In this context, it is necessary to establish a systematic that enables the development taking into account the principles of RAMS (Reliability, Availability, Mantainability 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 all types of systems. Today it has been adopted by governments such as the German [https://www.cio.bund.de/Web/DE/Architekturen-und-Standards/V-Modell-XT/vmodell\_xt\_node.html] or the United States for the development of transportation systems [https://ops.fhwa.dot.gov/publications/seitsguide/seguide.pdf] and road infrastructures [https://www.fhwa.dot.gov/cadiv/segb/views/process/]. It has also been standardized by IEC 62278 for the railway industry and ISO 26262 for the automotive industry. Annex 5 presents other life cycle management methodologies.

This model (Figure 2) 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 2 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 activies are perfomed 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 Cycle (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 2 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 2.

Finally, table X 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 of 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. |

|  |
| --- |
|  |
| Figure 3 Validation and verification activities mapping in V-model |

# Model-based design

## Models as the center for product development

## Tools for model-based development

Existen diferentes técnicas de simulación para verificación en base a modelos, como son el *Model-in-the-loop (MIL),* el *Software-in-the-loop (SIL),* el *Processor-in-the-loop (PIL),* el *Hardware-in-the-loop (HIL)* y el Prototipado Rápido de Control (*RCP*). A continuación, se detallan las características y estructura de cada una de estas estrategias para una aplicación de control clásica como la mostrada en la Figura 3.1.

|  |
| --- |
|  |
| Figura 3.1 Bloque diagrama de un sistema de control básico |

### Model-in-the-loop

En la técnica Model-in-the-Loop el sistema completo, tanto el control como la planta está simulada.

|  |
| --- |
|  |
| Figura 3.2 Estructura MiL |

Para el caso de Matlab/Simulink, se trataría de tener un modelo de la planta y un modelo del control. El modelo de control se refiere a una estrategia implementada por diagramas de bloques o máquinas de estados. En este escenario, el control no está todavía codificado. Lo habitual es una simulación pura en ordenador, pero pueden existir casos donde las simulaciones se hagan en simuladores en tiempo real.

### Software-in-the-loop

La técnica Software-in-the-loop consiste en la implementación del control en código. El producto sigue estando implementado en plataformas de testo, como un ordenador o un simulador en tiempo real, pero la implementación es real. El código se acerca a la versión final que irá embarcada en el hardware de control. Este escenario permite verificar el código, las secuencias de ejecución e incluso dimensionar la memoria necesaria en el futuro procesador en tiempo real.

|  |
| --- |
|  |
| Figura 3.3 Estructura SIL |

### Processor-in-the-loop

En este escenario, el control ya está implementado en un controlador hardware, aunque sea a modo de prototipo. De esta forma, se pueden medir los tiempos de ejecución, el uso de la memoria etc. En aplicaciones de control, dado que el controlador deberá leer o escribir señales analógicas o digitales, puede ser necesario que la planta se simule en un simulador en tiempo real. Esto exige que la planta se ejecute y de una respuesta por debajo del tiempo de ejecución mínimo del control.

|  |
| --- |
|  |
| Figura 3.4 Estructura PIL |

### Hardware-in-the-loop

Esta herramienta permite validar el hardware y el software de control sin necesidad de tener la planta real. El simulador dispone de las entradas y salidas analógicas y digitales para poder comunicarse con el controlador, para que este pueda ejecutar el software de igual forma que en la aplicación real. Dependiendo del controlador, en el simulador se ejecutará el modelo de la planta en tiempo real, incluso se pueden implementar comunicaciones de todo tipo para emular el sistema controlado real.

|  |
| --- |
|  |
| Figura 3.5 Estructura HIL |

### Prototipado rápido de control

En los casos en los que el sistema a controlar ya existe, esta técnica permite implementaciones experimentales de la estrategia de control en entornos de pruebas. Sin necesidad de tener el hardware de control final, se puede probar la estrategia de control en equipos de laboratorio para acelerar el proceso de verificación.

|  |
| --- |
|  |
| Figura 3.6 Estructura RCP |

## Model-based design and the V-model

La metodología mostrada en la **¡Error! No se encuentra el origen de la referencia.** es aplicable a multitud de sistemas. En el ámbito de sistemas de energía, este modelo en V se ha apoyado en herramientas de modelado y simulación. De este modo, no se espera hasta tener el producto final para verificar y validar el producto, sino que desde la fase de diseño se dispone de herramientas para testear los desarrollos, por muy prematuros que sean. Antes de tener componentes reales disponibles, los modelos pueden sustituir el sistema real y ayudar en la detección prematura de fallos de diseño. Se utilizan modelos, tanto para sustituir al sistema a controlar, como para tener una versión inicial del control.

SACAR DEL ESTADO DEL ARTE DE DAVID

Las herramientas de desarrollo basadas en modelos se utilizan en las diferentes fases del modelo en V (Figura 3.7).

|  |
| --- |
|  |
| Figura 3.7 Desarrollo basado en modelos y modelo en V |

Tomando como ejemplo el desarrollo de una unidad de control para accionamientos eléctricos (software + hardware), se suelen utilizar las herramientas descritas anteriormente de la siguiente forma:

* En las fases de diseño no existe ni software ni hardware de control. La técnica MiL permite simular y analizar la arquitectura y el diseño de cada subsistema de forma detallada.
* En la fase de testeo unitario, el hardware y el software se siguen probando por separado. De cara al software existen varias opciones. Mediante SiL, se integra el código de control en la simulación o mediante PiL se puede probar el código en una tarjeta de control de prueba, frente a una planta simulada en tiempo real. Mediante RCP se puede probar el código de control frente a la planta real (o de laboratorio) en un controlador de prueba.
* En la fase de verificación de subsistema, se utilizan las mismas herramientas que en la anterior, pero se pueden probar componentes en conjunto o subsistemas completos.
* En la fase de integración, el software y el hardware de control se integran y se verifican de forma gradual en HiL, laboratorio y escenario real.
* En la fase de aceptación y validación se utilizan las mismas herramientas para poder asegurar el cumplimiento de los requisitos del sistema.
* En la operación y el mantenimiento se siguen utilizando las herramientas mencionadas para resolución de incidencias o diseño de mejoras.

<https://es.mathworks.com/help/rtw/gs/v-model-for-system-development.html#brufb98-8>

https://es.mathworks.com/help/ecoder/gs/v-model-for-system-development.html#brufb98-7

# MATLAB & Simulink for V-model application

https://es.mathworks.com/help/ecoder/gs/v-model-for-system-development.html#brufb98-7

# Anexo: Modelos de gestión del Ciclo de Vida

### Waterfall

La metodología Waterfall (también llamada en cascada) fue una de las primeras que planteó la sistematización del desarrollo del Ciclo de Vida del software [Software development lifecycle models]. Tiene cinco etapas que se ejecutan de forma secuencial. Esto quiere decir que para empezar una fase la anterior tiene que estar terminada (ver Figura 5.1).

|  |
| --- |
| Figura 5.1 Metodología Waterfall [Managing the Development of Large Software Systems] |

In this model, the core of the proccess are the analysis and the coding steps. These steps can be found in any software development case, but in order to assure quality and fulfill customer requirements, requirements and testing steps are added.

La desventaja de esta metodología es que no refleja las diferentes iteraciones que pueden suceder entre las fases del proceso. En ocasiones es difícil mantener unos requisitos de principio a fin durante todo el proceso, puesto que pueden cambiar según se vayan encontrando dificultados. Incluso el diseño puede cambiar en función de los resultados de la fase de testeo. Por ello, Royce [Managing the Development of Large Software Systems] propuso una variante donde cada fase puede tener retroalimentación con la anterior o incluso con más de una fase anterior.

|  |
| --- |
| Figura 5.2 Metodología Waterfall [Managing the Development of Large Software Systems] |

Una de las aportaciones de esta variante es la definición exhaustiva de la documentación a generar en cada fase, como se muestra en la Figura 5.3. El autor destaca que la documentación debe ser la herramienta de comunicación principal entre los responsables de cada fase. Además, hace hincapié en que el valor tangible de la documentación se puede reconocer en las últimas fases del proceso:

* Durante la fase de testeo, tener una buena documentación sobre el diseño permite concentrarse en los fallos que pueda haber. Si no, la persona responsable del testeo pasa más tiempo intentando entender cómo funciona el sistema que testeándolo.
* Durante la fase de operación, con una buena documentación, el usuario se centra en el proceso operación del producto. Si la utilización es difícil de entender y no hay documentación, el que lo utiliza acaba siendo el mismo que lo ha diseñado, que a lo mejor no entiende tanto sobre el producto final.
* En posibles tareas de rediseño, actualización o retrofiting.

|  |
| --- |
| Figura 5.3 Metodología Waterfall [Managing the Development of Large Software Systems] |

Por último, cabe destacar que Royce también añade una fase intermedia entre el requisitado y el análisis, llamada diseño preliminar. En esta fase se asegura que los requisitos se van a poder cumplir con los recursos disponibles en las siguientes fases, evitando así que un diseño totalmente erróneo llegue a las fases de codificación y testeo.

### B-model

El modelo en B [A Practical Handbook for Software Development], es también una variante del Waterfall que tiene en cuenta las iteraciones que se pueden dar durante la fase de mantenimiento. Este modelo parte de la idea de que la mayor parte del gasto no se da en el desarrollo sino en la operación y el mantenimiento. Es más, contempla que el producto preliminar se puede mejorar y actualizar durante toda su vida útil. Por ello divide el modelo en dos ciclos: el de desarrollo y el de mantenimiento.

|  |
| --- |
| Figura 5.4 Modelo en B [A Practical Handbook for Software Development] |

A su vez, es interesante ver cómo se empiezan a proponer sub-procesos como el prototipado para ayudar en la definición y diseño del sistema. Los autores destacaron en su momento que el prototipado desde el punto de vista de hardware era una práctica común y muy provechosa, dado que podría ayudar en dar por buena una idea o concepto de producto sin necesidad de utilizar los materiales finales y con un coste relativamente bajo. Sin embargo, durante muchos años al prototipado a nivel de software no se le vio utilidad, por dos razones: se creía que el mismo prototipado no era rentable y no se disponía de las herramientas adecuadas. De todas formas, en [A Practical Handbook for Software Development] proponen un ciclo interno que define las fases de prototipado.

|  |
| --- |
| Figura 5.5 El ciclo de desarrollo complementado con un ciclo de prototipado[A Practical Handbook for Software Development] |

En lo que respecta a esta tesis, es interesante ver que el concepto de prototipado y la validación iterativa de soluciones ya se venía proponiendo desde finales de los años 80, lo que ha desembocado hoy en día en el uso extensivo de modelos y herramientas de simulación durante todo el ciclo de vida.

### Spiral model

El modelo en espiral tiene un enfoque diferente a los modelos presentados anteriormente. Mientras que el resto pone el foco en los requisitos y el desarrollo a partir de ellos, el modelo en espiral propone un proceso sistematizado y continuo de evaluación de riesgos. Como puede verse en la Figura 5.6, este modelo establece para cada fase cuatro tareas principales:

* Planificación de fases.
* Identificación de objetivos, alternativas y limitaciones.
* Evaluación de alternativas e identificación de riesgos.
* Desarrollo y verificación de productos.

|  |
| --- |
| Figura 5.6 Spiral model [A Spiral Model of Software Development and Enhancement] |

Con la progresión en espiral se establece que en cada fase (concepción, requisitado, desarrollo e integración y testeo) se realizan las cuatro tareas. Además, se contempla la posibilidad de que en cada fase exista un prototipo.

This model divides risks into two groups: development risks and performance risks. If performance risks are still predominant, the proccess continues to the next phase of the spiral in order to clarify how to improve the performance with the next prototype. If the risks are related with the development of the product, the classical Waterfall model is used to develop the producto in detail.

As it was mentioned before, this is a risk-driven model. Many manufacturers have two different proccesses: one for development and another one for risk analyisis and evaluation. This approach allows to reflect both of them in the same model.

# Referencias bibliográficas

[1] R. Kneuper, *Software Processes and Life Cycle Models: An Introduction to Modelling, Using and Managing Agile, Plan-Driven and Hybrid Processes*. 2018.