mathematics

MDPI

Article

A Continuous Process for Validation, Verification, and Accreditation of Simulation Models

Pau Fonseca i Casas

Department of Statistics and Operations Research, Universitat Politècnica de Catalunya-BarcelonaTech, 08034 Barcelona, Spain; pau@fib.upc.edu

Abstract: A simulation model, and more generically, a model, is founded on its assumptions. Assurance of the model's correctness and correct use is needed to achieve accreditation. Often the exercise of working with a specific code misunderstands the overall process, focusing the resources on the model coding and forgetting the needed resources to ensure the validation of every step of the model definition and coding. The goal of this work is to present a methodology to help in the definition and use of the assumptions in the modeling process. To do so, we present a process to conduct a simulation project, an assumptions taxonomy, and a method that simplifies working with those assumptions. We propose to extend the traditional Validation, Verification, and Accreditation processes to a process composed of eight Validation, Verification, and Accreditation phases that cover the overall life cycle of a model. Although this paper is focused on a simulation model, we can extend the proposed method to a more general modeling approach.

Keywords: validation; simulation; assumptions; verification; accreditation

MSC: 90-10; 00A71

check for updates

Citation: Fonseca i Casas, P. A Continuous Process for Validation, Verification, and Accreditation of Simulation Models. *Mathematics* **2023**, *11*, 845. https://doi.org/10.3390/math11040845

Academic Editor: Aleksandr Rakhmangulov

Received: 7 December 2022
Revised: 26 January 2023
Accepted: 28 January 2023
Published: 7 February 2023

1. Introduction, the System, and the Model

We want to start with a brief discussion regarding the system and the model. The system is the subject of our analysis; that can be something real, something physical that exists or not. On the other side, we have the model, the abstraction we will use to represent this system.

More specifically, a system is defined in the frame of systems engineering as "*a construct or collection of different elements that together produce results not obtainable by the elements alone*." ("Systems INCOSE-Systems Engineering-UALR") [1], while a model is "(. . .) a representation and abstraction of something such as an entity, a real system, a proposed system design or an idea" [2].

Therefore, following the proposal of [3], the main properties a model owns are:

1. Mapping: The goal of models is to represent something, i.e., mappings from natural or artificial systems or models.
2. Reduction: Models, in general, do not describe all the features of the original system, rather, only those features that seem important to modelers or system stakeholders.
3. Pragmatism: The models themselves are not explicitly assigned to their originals. They fulfill their surrogate functions (a) for certain cognitive and/or behavioral model-like subjects, (b) over certain time intervals, and (c) restricted to certain mental or practical processes.

These properties must be taken into consideration when one builds a model. Specifically, the process of building a simulation model is iterative, usually involving different personnel. This process starts by describing the system we want to represent, that is, the object of our analysis, and from it, we start a description of the key elements that must be considered to define the model. Once we understand what is "my system" we can go

further to describe the problem that we are trying to solve. A good thought is that "*all models are wrong* (. . .)", but some of them are useful [4]. Therefore, to ensure that we are applying a useful model, we follow a validation process since our model, although not completely correct, can be useful to our purposes: "*A simulation model should always be developed for a particular set of objectives. In fact, a model that is valid for one objective may not be for another*" [5].

The core of the model is its assumptions. They represent how we understand the system and express how we will conceptualize the model and, from this, how we develop the coding. In simulation [6,7], the Validation and Verification of the model's assumptions become key aspects to ensure the correctness of the conclusions obtained from a model. The model can be viewed as an experimental framework able to conduct new experiments [8], whereas wrong models can provide interesting insights [9] for both modelers and clients.

Validation, Verification, and Accreditation processes in simulation have been widely discussed, defining structures, diagrams, and recommendations that help the modeler ensure that the modeling process will succeed. One can review [10] for a simplified version of the model development process, and a more complex view developed by the same author can be found in [11]. The experimental design for a simulation model is detailed in [12], while [2] proposes a set of rules for the Validation, Verification, and Certification of simulation applications. The progress and challenges of a key element in the process of a description of a simulation model, the conceptual modeling, are discussed in [13].

In this work, we will present a pragmatic approach with practical use for modeling, allowing us to identify guidelines to follow in the process of defining a simulation model that must be accredited to eventually implement some solutions in the system. We discuss the current state of the techniques that can be used for the validation of the assumptions, presenting their integration with an improved cycle of a simulation model and showing their application in an industrial case. We will also discuss how this approach has an impact on the creation of a digital twin in the frame of Industry 4.0.

Also, in this paper, coded means *writing*: that is, implementing the program that is going to execute the model. Implementation means the modification of the system according to the accepted results. Validation means ensuring that the model is correct, while Verification means ensuring that the model has been correctly coded. Accreditation means that the stakeholders believe in the model and will use it to implement or modify (according to the model results) the system. We also want to mention that assumptions and hypotheses are often considered equivalent but are not. A hypothesis must be tested by an experiment, while an assumption can be accepted tacitly. In any simulation model, all the assumptions will be formulated through a conceptual model or a computerized model in a mathematical framework and must be assessed continuously. Therefore, we will be able to define an explicit method to evaluate the validity of these assumptions; hence, we will talk about hypotheses and not assumptions when explicit processes to test these assumptions exist. However, since we do not know if this process will exist in the development of our simulation model, we will use the term assumptions in this document for the sake of simplicity.

This paper is structured as follows. First, in Section 2, we review the existing approaches. Next, in Section 3, we present the process to define a simulation model, an iterative approach that generates different products. In Section 4, we present the assumptions taxonomy. In Section 5, we discuss how to validate those assumptions using the iterative cycle presented in Section 3. We present a table (W3H) that systematizes the selection of a subset of the tests one can use to perform the validations; moreover, we discuss the utility of work with non-validated assumptions. In this section, we also present the assumptions table, a product that helps in the model conceptualization and assumptions use and maintenance. In Section 6, we present an example based on a real project developed in the context of the definition of a Digital Twin for a factory. Finally, some conclusions are drawn in Section 7.

2. Existing Approaches

The classic Software Development Life Cycle (SDLC) is composed of four stages (i) planning; (ii) analysis; (iii) design, and (iv) implementation [14]. From this, we can note that several different Software Development Models exist, such as Waterfall, V-Model, RAD model Iterative model, Agile, Spiral, and Prototyping models, among others [15,16].

In the case of a simulation model that will end up in the coding of a computer program, we will conduct similar phases for the implementation and verification of the simulator. This process will be guided continuously through the Validation, Verification, and Accreditation (VV&A) processes. This guidance is because of the specific particularities of modeling and due to the experimental nature of the results that must be implemented in the system. Therefore, during the process, it is necessary to ensure that not only the computer program fits well with the conceptual model but also that the conceptual model and the assumptions they represent are correct enough to obtain valid conclusions regarding the system [10]. In the case of a simulation model, the system (our object of the analysis) is not necessarily a piece of software, but a factory, the environment, a society, or any other system. Therefore, the VV&A process guides the complete development of a model to ensure that all the assumptions we will use in the model development are correct.

Several proposals exist to conduct the VV&A cycle in a simulation project; [10] presents an approach to ensure the validity of a model using a simplified cycle that presents the key elements of the simulation model building. A more complete vision is presented in [11], including a distinction between the real world and the simulated world. There is a distinction between the "System", "System Theories", "Conceptual Model", "Simulation Model Specification", "Simulation Model", "Simulation Model Data/Results", and the "System Data/Results", composing a process that conducts the modeling process. Another proposal for a detailed cycle for modeling and simulation is presented in [17]. It encompasses the different elements that one can find in a simulation model like documents ("Formulated Problem", "Requirements Specification", "Conceptual Model", "Architecture Specification, "Design Specification", "Executable Submodels", "Simulation Model", "Simulation Results" and "Presented Results"). This proposal is categorized, according to [18], with the different phases where a simulation model could be: the Problem and Requirement phase, the Design and Programming phase, the Simulation and Certification phase, and the Storage and Reuse phase.

Coherently in [5], assume that Validation, and specifically Data Validation [11], owns a central role in the VV&A process. Therefore, a proper understanding of the nature of the assumption becomes relevant. In [19], the authors provide a taxonomy for modeling assumptions, hypothesizing them, and proposing a use in simulation experiments. They argue that depending on how the assumptions are constructed, one can classify them as either constructive or non-constructive and phenomenological or mechanistic, depending on the abstraction level. With this initial classification, the proposed taxonomy classifies hypotheses as (i) phenomenological hypotheses, which should make statements about the relationship between the simulated inputs and outputs; (ii) mechanistic hypotheses that define the theoretically grounded model's fundamental mechanisms that produce and maintain the desired behavior, and (iii) control hypotheses related to the optimization of behavior. In [20], the authors provide a detailed process for conducting hypothesis-driven simulation studies and show how such explicit hypotheses models can facilitate validation processes. Both [21,22] present a more general approach, describing the phases of the life cycle of a simulation study based on the use of assumptions. These studies try to accommodate the structure of the assumptions used in a simulation model but not by outlining how one can perform a validation of the model assumptions.

However, currently, there is no complete view of the overall process that combines the description, classification, and use of the model assumptions and how to manage them in a detailed Validation, Verification, and Accreditation process. This will help to detect what tests can be applied to each assumption to increase the credibility and accountability of the models. Moreover, there is no description of the impact of the Digital Twin concept

in the context of this process. In this work, we propose an approach that, from a very pragmatic point of view, allows for an understanding of the nature of the assumptions we use and how to use them in a complete VV&A life cycle that also encompasses the Digital Twin concept.

3. Defining a Simulation Model

The complete VV&A cycle that guides the development of a simulation model by introducing all aspects that must be validated in the cycle of a modeling project is presented in Figure 1. With the emergence of the Digital Twin concept (in the frame of Industry 4.0), some validations, like "Solution Validation", emerge as a key element to ensure the correctness of the proposed solution and the implementations completed in the system. This allows us to define the concept of model-based discussion that starts from the software engineering concept of Model-Based Systems Engineering [23]. This is in line with the hypothesis-driven simulation studies as proposed in [24], the formal specification hypothesis [21], and also the goal [19]. Notice that due to the cyclic nature of modeling, different versions or prototypes can appear. This will be a source of discussion to understand the validity of the assumptions by performing an analysis between different assumptions used on the different model versions.

The *Goals*, that is, why we are building the model, determines the definition of the *Problem Entity* that defines the assumptions that are going to be used on the project and formalized with the *Conceptual Model*, along with the *Scenarios/Configurations* definition. Although the goals have a clear impact on the *Problem Entity* and consequently on the *Conceptual Model* and the *Scenarios/Configurations* of the model, they rule the overall development of the project. As an example, the selection of the software (or tools) that must be used to do the model coding and the subsequent experimentation with the model must be guided by a clear definition of the goals that will be translated into a set of metrics that allow this selection following a proper methodology [25,26]. Therefore, depending on the goals, the software to be used to code the model must own some specific features or others.

Each one of these products can end up with different documentation. Here we propose to define a document for each of these products, as is presented in Table 1. This documentation allows presenting the product, but also it will be present the process of its development; hence, these documents are living documents updated continuously along with the life of the system and the model they represent.

To transform one product into another product, we have the **actions**. They are (i) *Problem Definition*, (ii) *Analysis and Modelling*, (iii) *Coding*, (iv) *Parametrization*, (v) *Calibration*, (vi) *Experimentation*, (vii) *Understanding of the solutions,* and (viii) *Implementation*. All these actions finish with a product that is going to be enough for our purposes depending on the validation processes that must be done continuously.

The different **validation processes** one must do are the: (i) *Conceptual Model validation*, which allows detecting if the definition of the model is complete and the model allows to answer the questions proposed by the Goals; (ii) *Data validation*; (iii) *Verification* of the model coding; (iv) *Operational validation;* (v) *Experimental validation;* (vi) *Accreditation,* and finally (vii) *Solution validation*. These validation processes can be done depending on the status of the development of the model. Notice that the definition of the goals also implies a validation, (viii) *Goals Validation*, related to the system management, which will be focused on assuring that the goals are aligned and detailed completely and correctly, with the main purposes of the system stakeholders.

As modeling is an iterative process, it generates several prototypes during the different iterations. In each of these iterations, the assumptions can be modified according to the goals and the evolution of the several products that answer those goals, driven by the actions and the validation processes. The iterative nature of modeling makes it natural to use an approach to coding like Agile, Prototyping, Iterative, and Incremental or also Rapid Application Development (RAD) methodologies.

Before entering more details regarding the different validation processes to be done, in the next section, we discuss the assumptions of the model and how a taxonomy of those assumptions helps in the different validation processes that must be done to ensure the correctness of the products.

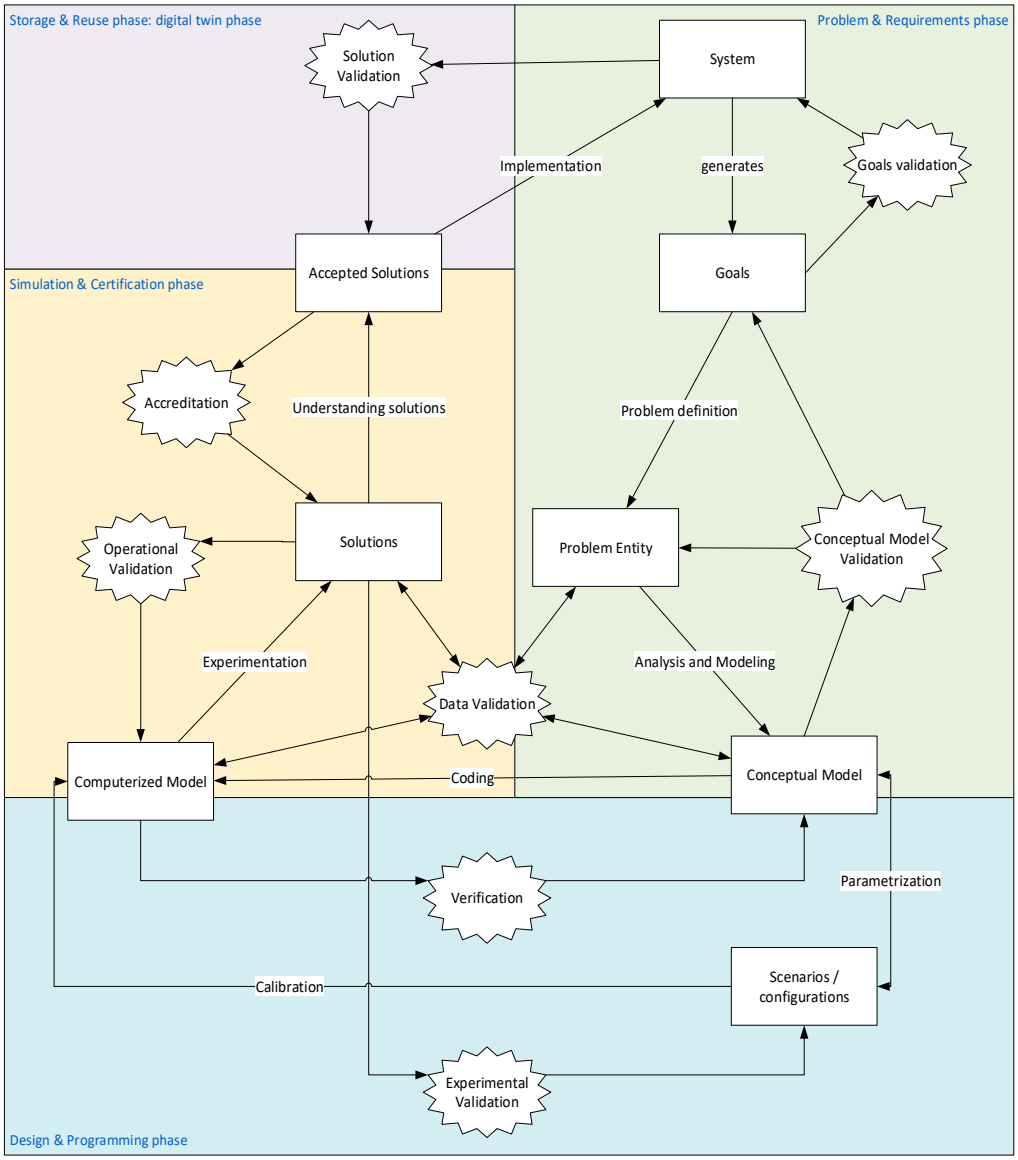

Figure 1. The proposed life cycle for a simulation project. The squares represent products, like the definition of the problem, the model coding, or the solutions obtained. The stars represent validation and verification actions that must be done to ensure the correctness of the products. If this process fails, we will come back to the previous product. In order to navigate from one product to another, an action must be done, like the coding to obtain the "Computerized Model" from the "Conceptual Model". On the diagram, the cycle is mapped on the different phases proposed by [19]. In Figure 1, the various products that are going to be affected or created by the simulation project are represented in a square. There are eight of these: (i) the *Goals*, since the clarification of the relevant elements that rule the organization are a product by itself; (ii) the *Problem Entity*, which is the definition of the problem that we are going to analyze that is generated by the *Goals* of the organization; (iii) the *Conceptual Model*, that represents a complete and unambiguous representation of the model; (iv) the *Scenarios/Configurations* to be analyzed; (v) the *Computerized Model*, that is the simulator that codes the model; (vi) the *Solutions*, (vii) the *Accepted Solutions*, accepted because a model-based discussion, and the (viii) *System* itself since it will be modified based on the *Accepted Solutions*.

Table 1. Proposed documents were obtained in each product of the modeling cycle.

PRODUCT	DOCUMENTATION	EXPLANATION
SYSTEM	System documentation	All the documentation regarding the system operations. Tacit and informal information can also be used here to understand the system's true behavior.
GOALS	Goals document	A document that contains the goals that drive the development of the model.
PROBLEM ENTITY	Assumptions document	A document that contains the assumptions of the model.
CONCEPTUAL MODEL	Conceptualization of the model	The conceptual model constitutes a specification and includes all the assumptions (not the simplifications) detailed in the Hypotheses document.
SCENARIOS/ CONFIGURATIONS	Experimental design definition	A document containing the experimental design. A discussion regarding the quality of the RNG used in the experimentation can be added here if needed.
COMPUTERIZED MODEL	Code and technical documentation of the code	Along with the code, the technical documentation explains the code and the needed configurations of the computational infrastructure needed to execute and maintain the model.
SOLUTIONS	Solutions document	Description and data containing the solutions obtained from the model's execution.
ACCEPTED SOLUTIONS	Executive summary of the project	Conclusions of the modeling project and suggestions to perform the implementation on the system.

4. Working with the Assumptions

Systems can be complex realities. Even though when we can work with simple systems, we need to use assumptions to describe what the system is. Once it is clear what the system is, we must define a model that represents and connects all the assumptions through the model **structure** and the **behavior**.

This model must be conceptualized using a formalism independent of the selected implementation tool. The tool must be selected depending on a clear set of metrics ruled by the goals and never by any other preference [25]. Several books and papers detail the advantages of using a conceptual model. Here, we want to focus on three aspects. First, the model conceptualization can be considered a product by itself [27]. We support this, as represented in Figure 1, because the knowledge representation that rules the different processes in a system can sometimes be more interesting than the simulation by itself. The conceptual model transforms the tacit knowledge of the system into explicit knowledge that can later be used to make decisions. A formal representation of a system, through the model conceptualization, helps to understand how the system behaves.

Second, a formal model representation simplifies its coding and enhances its maintainability. Third, a conceptual model simplifies the model understanding by the different actors that participate in the simulation project. This improves the communication between diverse profiles simplifying the validation processes, a key aspect of the IT/OT convergence (Information Technology and Operations Technologies) in the frame of Industry 4.0.

Since the conceptual model is founded on assumptions, the process of validating a model is the process of validating the different assumptions used to build the formal representation of the simulation model and accepting the needed assumptions as valid. The conceptual model starts with the definition of the assumptions document. This document represents how we understand the behavior and the structure of the *System* under the lens of the *Problem Entity*: the reason we build the model. The conceptual model is the formal representation of the assumptions we are going to validate in the process of building the model. However, it is first necessary to detail the nature of the assumptions. An initial proposal of this taxonomy is presented here [28], defining three main types of assumptions: Systemic Data assumptions, Systemic Structural assumptions, and Simplification assumptions.

Assumptions Taxonomy

First are those assumptions that allow us to define how the system behaves. If we have a deeper knowledge of the system, more assumptions describing its behavior can be used. We want to add as many as we can of these assumptions (if they are useful for the model goal) since they help us in the description of the model. We suggest naming these assumptions **Systemic** since they describe the behavior of the system. These assumptions represent our current knowledge of the system and are assumed to be true (but may not be true). These assumptions can also be divided into two categories.

(i) **Systemic Data assumptions** are those related to the data, for example, those that define the probability distributions that represent the model element's behavior.

(ii) **Systemic Structural assumptions** can be those that represent the relations between the different elements that compose the model, the model behavior, and the causal relations between the elements.

Second, there is a category composed of assumptions that simplify the model we are going to build.

(iii) **Simplification assumptions** are useful to reduce the complexity of the model. Because of the resources, the time we own to code the model, and the limited knowledge we have of the system, we must always use these kinds of assumptions. They represent our current limitations in two directions: (i) system knowledge or (ii) resource knowledge and are assumed to be false (but may be true, becoming Systemic). We want to keep the model small enough to be useful, keeping the model simple but not simplistic. Notice that in the hypothetical case that we have a complete understanding of the system and infinite resources, there is no need to use simplification assumptions in our models.

This taxonomy has the main goal of simplifying and helping in the VV&A process and informing the understanding of the nature of the decision taken during the modeling process. They also represent the current limitations and understanding of our system; hence, as we will show next, it will be interesting to store this information in a structure named the assumptions document.

As an example, considering a linear model, one can introduce an error (i) when omitting a relevant variable (because we do not know it, in that case, this is a Systemic Structural assumption error, or because we assume that it is not relevant, in which case we are using a Simplification assumption erroneously); (ii) when including an irrelevant variable (using a false Systemic Structural assumption), or (iii) when specifying a linear relation that does not exist (again, using a false Systemic Structural assumption). The meaning of the assumptions is the key to interpreting the causality of the model and performing an accurate Validation. In addition, following the example of a linear model, the hypotheses regarding the data (independency, normality, homoscedasticity ...) are Systemic Data assumptions that must be satisfied. As we will see next, we can use a model that does not have all the assumptions validated if we clearly understand our model goal and what is the nature of the assumptions that fail.

5. Validating a Simulation Model

Validated means that the Systemic assumptions are tested (or assumed) as true, and the Simplification assumptions are tolerable by all the parts involved in the modeling project and supported by the results. However, we cannot ensure that a model obtained from a set of assumptions is true (we can only ensure that a model is false [29].) but useful for our goals. Modeling can also be useful for many other reasons, instead of predict [30], we can assume that a model is valid for a specific purpose (but false for others), implying that the underlying assumptions are valid for this purpose.

Hence, what do we mean by Validation? Validation can be seen differently depending on the view used to understand the assumptions [31], with the following being the more common approaches [32]: (i) the rationalist view (where the model becomes a structure

that starts from premises that act as axioms); (ii) the empiricist view (where the model must be tested empirically with experimental data), and (iii) positivist view (where a model is going to be considered valid only if accurate predictions are going to be obtained).

To conduct the Validation process in a simulation project, Naylor and Finger [32] proposed combining the three historical methods of rationalism, empiricism, and positivism into a multistage process of validation, the multistage or utilitarian approach. This validation method consists of [32]:

(i) Developing the model's assumptions, observations, and general knowledge.
(ii) Validating the model's assumptions and testing them empirically if possible.
(iii) Comparing the input-output relationships of the model with the real system.

Of the different processes we propose in Figure 1 that one must conduct to ensure the correctness of the solutions implemented in the system, the essential steps that must be completed are:

1. **Data validation** to ensure that the data we are going to use on the model, the probability distributions, the data sources that are used to obtain the data, the support structures to keep the data, etc., are accurate and correctly defined. To validate the data, we must focus our efforts on the Systemic Data assumptions. The tests must focus on analyzing the data we are going to use in the model in two directions: (i) assuring that the data fits well with the system process it wants to model (data fitting) and (ii) ensuring that the data expiration constraints are valid, therefore, guarantee that the institutions (structures, enterprises, data warehouses, etc.) needed to obtain the data works and keep this data valid during the life of the model. Defining data expiration constrain is a key aspect to ensuring the validity of the simulation models (that eventually will be part of a Digital Twin in the frame of Industry 4.0) since the model is going to be a representation of a real system, analyzed along with the life of the system that emulates.
2. **Conceptual model validation** is determining (i) the correctness of the conceptual model assumptions and (ii) that the problem entity model's representation, the model's structure, and behavior, represented by the logic and the mathematical and causal relationships, are "reasonable" for the goals. To validate the conceptual model, we must focus on the formal representation of the model or on the coding of this formal representation to validate the Systemic Structural and Simplification assumptions. The conceptual model represents the structural relations of the different model elements and the behavior of the different elements that compose the model. This implies that we must define techniques to ensure that (i) the Systemic Structural assumptions are correct for our purposes and (ii) the Simplifications assumptions do not transform the model into a simplistic version of the reality it represents. Several formal languages like Specification and Description Language (SDL) [33–35], Petri Nets [36–39], and DEVS [40–42] allow an analysis of its behavior before the coding (reachability analysis, etc.). Moreover, there are methods to transform a conceptual model from one to another, becoming excellent tools to ensure that the underlying assumptions are correctly defined [43,44]. This also simplifies the integration between different conceptual models that express the same system or different parts of a system that must be defined as one following meta-formalism, common formalism or co-simulation approaches [45,46].
3. **Operational validation** is focused on the results that we can obtain from a model implementation. Some of the methodologies that we present next, like Black Box validation, among many others, imply the use of all the model assumptions since the modeler tries to validate the whole behavior of the computer program that codes the model. In that case, we often cannot distinguish if the results (in case the results are wrong) are due to incorrect Systemic assumptions (Data or Structural), due to a wrong Simplification assumption, or an incorrect model coding, introducing the verification phase into account. However, Operational Validation is one of the most

widely used approaches to do validations and must always be done to ensure that all the connections between the distinct parts have been correctly defined.

4. **Verification** ensures that the code has been completed correctly. The techniques at this point are not focused on understanding if the assumptions are correct, but if they have been correctly represented in the computational framework, we use them to implement the model.
5. **Experimental validation** analyzes if the experimental procedures used to obtain the results are good enough. A simulation model needs to be executed based on an experimental framework; hence the design of experiments and the execution of those experiments must be validated. One must ensure that the factors needed and the levels to be analyzed are correctly defined for the goals proposed. We must also ensure that the number of replications for each scenario in the experiment has been correctly obtained (along with the methods used to obtain these replications and the length of the replications). Without this validation, the results of a model cannot be considered for its implementation in the system.
6. **Solution validation** focus on the accuracy of the results obtained from the model proposed solution and the data obtained on the implemented solution. This is a key validation in the frame Industry 4.0 Digital Twin concept since it ensures that the model and the system are close enough to be useful for the goals proposed. Solution Validation is a useful validation for the modelers that will eventually learn about the divergences between the model proposed solutions and the implementation of the accepted solutions on the system. This will allow further modeling improvement through the detection of errors. It is also useful for the owners of the system to understand if the different validation processes, specifically those related to the accreditation, are working properly. Notice, however, that because this validation can show flaws in the accreditation process, in the model validation and verification processes, or during the system implementation, this validation is sensible and often forgotten. However, in the frame of cyber-physical systems (in Industry 4.0), this validation becomes a key element since the Digital Twin must accurately represent the current system, which is prone to change. This solution validation traditionally has been used to test if the model can also be used as a product, often to test the behavior of the system in other scenarios, see [47,48]; in the context of Industry 4.0 is a continuous process to ensure that the simulation model is yet valid. The nature of techniques one can use in the Solution Validation process and the assumptions that are going to be involved are similar to the ones used in the Operational Validation process.

To detect the assumptions that are validated depending on the method used, we propose a classification in the next section of some of the well-known validation methods and what the assumptions are that they try to validate. This is not an extensive list of the methods that can be used, but just a starting point to define a taxonomy that systematizes the overall Validation, Verification, and Accreditation process in a model.

5.1. Testing the Model Assumptions

In [2,5,11,49–53], one can review a description of the different tests one can apply for simulation validation. In Appendix A.1 VV Tests, we present a description of those tests following the proposed method of this paper. However, since the resources we have are finite, we must establish some classification to select the more appropriate test to be used. In [54,55], is presented a classification of the tests one can apply to a model according to the formal, informal, static, or dynamic nature of the techniques. We adapt this table using the Informal/Formal and Static/Dynamic categories as dimensions. With this idea, we propose Table 2. Informal tests are those tests where a high range of subjectivity exists because they are based on human perception. Informal tests require the intervention of specialists. Hence in this type of test, specialists are a needed resource. Static tests are those tests that do not need the execution of the simulation model to be applied. Static tests will always be focused on the model structure, while dynamic tests can be focused on the model

structure but also on the output data we obtain from this execution. Dynamic tests need the coding of the simulation model to be conducted. Formal/Dynamic tests can need real data (focused on testing the model logic) or can use fake data (focused on testing model data, mainly its output). This generates a subcategory, as is presented in Table 2.

Table 2. Classification of different validation tests, depending on their nature.

	STATIC	**DYNAMIC**	
INFORMAL	✓ Face Validation ✓ Walkthroughs ✓ Audit ✓ Desk Checking ✓ Documentation consistency checking ✓ Inspections ✓ Reviews ✓ Traceability Assessment ✓ Interface Analysis (model interface; user interface)	✓ Visualization/Animation ✓ Turing Test ✓ Graphical Comparisons ✓ Acceptance Testing ✓ Object-Flow Testing ✓ Compliance Testing (authorization; performance; security; standards) ✓ Product Testing ✓ Alpha Testing ✓ Beta Testing	
FORMAL	✓ Induction ✓ Inference ✓ Logical Deduction ✓ Lambda Calculus ✓ Predicate Calculus ✓ Predicate Transformations ✓ Proof of Correctness ✓ Cause-Effect Graphing (State Transition analysis . . .) ✓ Goodness-of-fit ✓ Data Analysis (data dependency; data flow) ✓ Control Analysis (state transition; calling structure; analysis of concurrent process; control flow) ✓ Semantic Analysis ✓ Structural Analysis ✓ Syntax Analysis ✓ Fault/Failure Analysis	✓ Partition analysis ✓ Path Analysis ✓ Symbolic Execution ✓ Assertion Checking ✓ Inductive Assertions ✓ Fault/Failure Insertion Testing ✓ Interface Testing (data; model; user) ✓ Special Input Testing (boundary value; equivalence partitioning; degenerative; extreme input; invalid input; real-time input; self-driven input; stress; trace-driven input; fixed values)	**TEST MODEL LOGIC**
		✓ Black-Box Testing ✓ Bottom-Up Testing ✓ Top-Down Testing ✓ Debugging ✓ Execution Testing (monitoring; profiling; tracing) ✓ Field Testing ✓ Predictive Validation (historical data) ✓ Regression Testing ✓ Sensitivity Analysis ✓ Statistical Techniques ✓ Sub-model/Module Testing ✓ Symbolic Debugging ✓ Structural (White-Box) Testing (branch; condition; data flow; loop; path; statement) ✓ Comparison Testing ✓ Event Validity	**TEST MODEL OUTPUT DATA**

Using this table as a starting point, we go further, analyzing **what** assumptions will usually be tested in each test and **when** (the validation process) the test can be applied. This classification is intended to be used as an approximation to understand when applying a specific test.

5.2. W3H Testing Table

Table 3 details **when** to do the test, **what** test to do, **how** to do the test (if the test is Dynamic, needs the model execution, and if it is Informal, needs the experts on the system participation), and **why,** that is the nature of the assumptions that will be tested. The final goal is to test whether all the assumptions are valid or useful for our model. Following this table, if we want to validate our simulation model, we need to apply at least one test to each assumption. This implies the selection of the tests that allow doing this better if we can select tests that are strongly focused on specific assumptions rather than all assumptions to allow us to detect possible sources of error.

Table 3. The final classification of the assumptions validated in each test depends on how the team is implementing the tests, but this table helps in the understanding of where to put the effort in the validation process. Finally, all the different validation processes must be done, and ideally, all the assumptions must be tested through the different validation phases.

WHEN	WHAT	HOW		WHY		
		Formal (F)/Informal (I)	Static (S)/Dynamic (D)	Systemic Structural	Systemic Data	Simplification
CONCEPTUAL MODEL VALIDATION	Audit	I	S	+		+
	Cause-Effect Graphing	F	S	+		+
	Desk Checking	I	S	+		+
	Documentation consistency checking	I	S	+		+
	Face Validation	I	S	+		+
	Induction	F	S	+		+
	Inference	F	S	+		+
	Inspections	I	S	+		+
	Interface Analysis	I	S	+		+
	Lambda Calculus	F	S	+		+
	Logical Deduction	F	S	+		+
	Predicate Calculus	F	S	+		+
	Predicate Transformations	F	S	+		+
	Proof of Correctness	F	S	+		+
	Reviews	I	S	+		+
	Traceability Assessment	I	S	+		+
	Walkthroughs	I	S	+		+
DATA VALIDATION	Events	F	D		+	
	Goodness-of-fit	F	S		+	
	Statistical Techniques	F	D		+	
VERIFICATION	Control analysis	F	S	+		
	Data Analysis	F	S	+		
	Debugging	F	D	+		
	Execution Testing	F	D	+		
	Fault/Failure Analysis	F	S	+		
	Interface Testing	F	D	+		
	Semantic Analysis	F	S	+		
	Structural Analysis	F	S	+		
	Symbolic Debugging	F	D	+		
	Syntax Analysis	F	S	+		

Table 3. *Cont.*

WHEN	WHAT	HOW		WHY		
		Formal (F)/Informal (I)	Static (S)/Dynamic (D)	Systemic Structural	Systemic Data	Simplification
OPERATIONAL VALIDATION	Assertion Checking	F	D	+		
	Black-box Testing	F	D	+	+	+
	Bottom-Up Testing	F	D	+	+	+
	Comparison to other models	F	D	+	+	+
	Compliance Testing	I	D	+		
	Degenerative	F	D	+		
	Extreme conditions	F	D	+		
	Fault/Failure Insertion Testing	F	D	+		
	Fixed values	F	D	+		
	Graphical Comparisons	I	D	+	+	+
	Historical data	F	D	+	+	+
	Inductive Assertions	F	D	+		
	Object-Flow Testing	I	D	+	+	+
	Partition analysis	F	D	+	+	+
	Path Analysis	F	D	+		
	Sub-model/Module Testing	F	D	+	+	+
	Symbolic Execution	F	D	+		
	Top-Down Testing	F	D	+	+	+
	Turing tests	I	D	+	+	+
	Visualization/Animation	I	D	+	+	+
	White-Box Testing	F	D	+		
EXPERIMENTAL VALIDATION	Sensitivity analysis	F	D	+	+	
ACCREDITATION	Acceptance Testing	I	D	+	+	+
	Alpha Testing	I	D	+	+	+
	Beta Testing	I	D	+	+	+
	Product Testing	I	D	+	+	+
SOLUTION VALIDATION	Field Testing	F	D	+	+	+
	Regression Testing	F	D	+	+	+

Notice that simplification assumptions are not validated in the strict sense. Since they are not introduced in the model, they represent the limits of our knowledge or the resources we own. This is the reason we do not detect a single validation method that is focused only on the Simplification assumptions; doing so implies the modification of the model, usually increasing its complexity and adding new Systemic assumptions to discard the simplification.

5.3. Working with Not Validated Assumptions

Validate a simulation model is a time-demanding and iterative task: see Figure 1. Achieving a fully validated model also needs a growing (exponential) number of resources [56]. In our path to find a useful model, we must often accept working with non-validated assumptions at some point to advance the model development. Table 4 shows if this situation is wanted, unwanted or useful because of the expected model results. The desired situation for an assumption is represented by "wanted"; "useful" can be transitory of a final situation for an assumption (consider it at the end of the completion of one of the sub-products following our SDLC) that can be needed for the model construction, and "unwanted" are those states that are not desirable for any purpose of the simulation

model, hence are just transitory states. We want to mention that the use of Simplification assumptions can be useful but never wanted. The final objective of a model is to work with as few simplifications as possible (or without simplifications). As we noted previously, simplification assumptions will always be false; we know that reality is more complex than the model.

Table 4. Effects of using no validated assumptions.

	Systemic Structural Assumptions	Systemic Data Assumptions	Simplification Assumptions
Validated	Wanted	Wanted	Useful
Non-validated	Unwanted	Useful	Unwanted

Systemic Structural assumptions represent the relations between the different elements that compose the model. If these relations are not well-defined, the model is not correct: the causality is the crux of a simulation model [57]. For that, using no validated Systemic Structural assumptions is an unwanted state.

For Systemic Data assumptions, using no validated data can be useful for testing purposes and to allow advance in the model definition without waiting for a dataset that needs time to be defined, prepared, etc. In some cases, it is necessary to use fake data to validate the behavior of the model (as an example in extreme conditions tests). Therefore, in these contexts, the use of no validated Systemic Data assumptions can be useful.

Finally, if we have validated Simplification assumptions in our model, we suppose their utility to accomplish our expected result considering the project constraints (technology level, budget, time, computational resources, knowledge, etc.). If these simplification assumptions are not accepted, that implies that we are using some simplifications in our model that the Stakeholders cannot assume. This situation is dangerous for the project, and often reflects bad communication with the client. Communication with the client from the beginning of the project and the definition of a good assumptions document are key elements for the success of a simulation project [5]. Again, note that working with all the assumptions in the "wanted" state is (often) unrealistic since this implies avoiding the use of simplification assumptions.

From this table, we can understand that the status of an assumption can be **true**, **valid** (when it is not true but is useful), and **false**.

5.4. Working with this Taxonomy, Writing the Assumptions Table, and Building the Conceptual Model

For the success of a simulation project, the assumptions document is a key element [5]. This document can start with some initial meetings between the Stakeholders and modelers and describes the model assumptions in natural language. This document must be simple and clear and, as we will show next, defined as a structure. In the frame of Industry 4.0, we propose to transform this document into a structured table that can be stored in a database, with an identifier that allows us to automatically detect whether a specific assumption is valid or not and where these assumptions are used on the model. We will also refer to a condition for the assumptions that allow understanding if these assumptions need revision, for example, the Systemic Data assumptions that have their own expiration date.

However, to accurately describe and use the Systemic assumptions, we must use formal language. This allows us to describe the structure and the behavior of the model completely and unambiguously. A formal language like SDL, DEVS, or Petri Nets, among others, becomes an excellent tool to represent the Systemic Structural assumptions and perform the conceptual model Validations. Hence along with the assumptions table, we must take care to keep the conceptual model representation of the system up to date to reflect our model.

The structure of the assumptions table must include the following:

1. **Type** of the assumption: Systemic Structural assumptions, Systemic Data assumptions, or Simplification assumption.
2. **Identifier**, which uniquely describes the nature of the assumption: as an example, SD_01 refers to the Systemic Data Assumption number "1".
3. **Description** of the assumption.
4. **Location** on the conceptual model, list of the diagrams that are affected by the assumption.
5. The validation **status** is true if the assumption is assumed correct by the Stakeholders or false otherwise.
6. **Review condition**, a function that modifies the Validation Status to false when this assumption must be reviewed: never (implying that the function is true), by time, or by a more complex set of rules. Systemic Data assumptions often have a data expiration value and hence must be reviewed at this time. In addition, Systemic Structural assumptions can change (because of changes in the system). For the Simplification assumptions, the introduction of a new technique means that we can now consider adding more detail to the model (or eliminating detail).

A Review condition is a necessary element when we model a digital twin (in the frame of Industry 4.0). In that case, the system and the model can follow separate paths, which leads to the need for the revision of the assumptions to keep the model useful. Notice that the assumptions table is a product by itself, and a company may be interested in keeping a digital version of this table to understand the assumptions that rule the behavior of the main company processes, to be used in conjunction with the model conceptualization and the code of the simulation model (that eventually will be used as a part of a digital twin).

As the construction of a simulation model is an iterative task, once we achieve an iteration where all the assumptions of the document have been accepted, we have a simulation model that eventually can be used to make decisions and can be prepared for Accreditation.

6. A Real Case Application Example

Now we are going to show how we will apply this to a real case: see [58]. The project was developed for a real industry that wants to optimize the behavior of a specific area of a production plant. Specifically, we want to perform a detailed simulation of the conveyor that moves some material from one point to another point of the factory. Note that this is not a short conveyor, and its behavior is critical for the good performance of the factory; since it is composed of several sections, it is a complex element that must be accurately represented. We want to configure the conveyor behavior, speed, and capacity to properly configure the machines located at the beginning and the end of the conveyor. The conveyor is connected with other elements, specifically in this example, with a machine named ADP, which provides the input material that the conveyor transports to a robot that will process them to other areas in the factory.

To do so, we talk with the managers to understand the main assumptions of the system, perform the classification, and define the assumptions document. From this, once the different Stakeholders agree on the correctness of the Structural assumptions and the suitability of the Simplification assumptions (to avoid unwanted states), we define a conceptual model based on Specification and Description Language (SDL) and ITU-T formal and graphical language. Note that the initial stages include Systemic data assumptions that have not been validated and some Simplification assumptions that may or may not be making the model too simplistic.

Following the cycle proposed in Figure 1, we must start with a clear definition of the goals. Once this is completed, the assumptions table must be defined, and the different validations are going to drive the achievement of the goals. If validation fails (see Figure 1), we go back to the previous product, revise it, and build again an updated version of the product that fails. The process will drive us back until a product supports the validations that must be done to test the assumptions, and we can complete the actions to build a new product that eventually is going to pass the Validations. The assumptions taxonomy helps

to understand the assumptions to be tested in each phase (see Table 3), but sometimes, not all the assumptions can be tested due to the incremental behavior of the modeling process. In our approach, we always present the assumptions table, and on this table, those in red are assumptions that are not yet validated but that we are going to use (considering them valid) to continue with the prototyping modeling process. In order to do so, an identifier for each assumption is proposed as an example. SD_01 is a Systemic Data assumption, SS_01 a Systemic Structural assumption, and SM_01 a simplification assumption.

This identifier, used for each assumption, allows us to later connect the assumptions table with the formal representation of the assumptions done in a formal language (SDL in our case). As an example, Figure 2 shows the first page of an SDL conceptual model for a conveyor element: see the details of the model [58]. The red squares represent those assumptions that are described in the assumptions document. See Table 5. The table contains the TYPE of the assumption, the IDENTIFIER of the assumption to simplify its use, the DESCRIPTION that describes the assumption, the LOCATION that details the element (or elements) of the conceptual model that receives the assumption, the STATUS of the assumption, that can be true, false or valid, and the REVIEW CONDITION that expresses when the assumption must be reviewed again to ensure that its STATUS did not change. Note that the conceptual model formally represents all the assumptions. Hence if all of the organization members are familiar with the language selected, this can be the communication channel to be used in the organization.

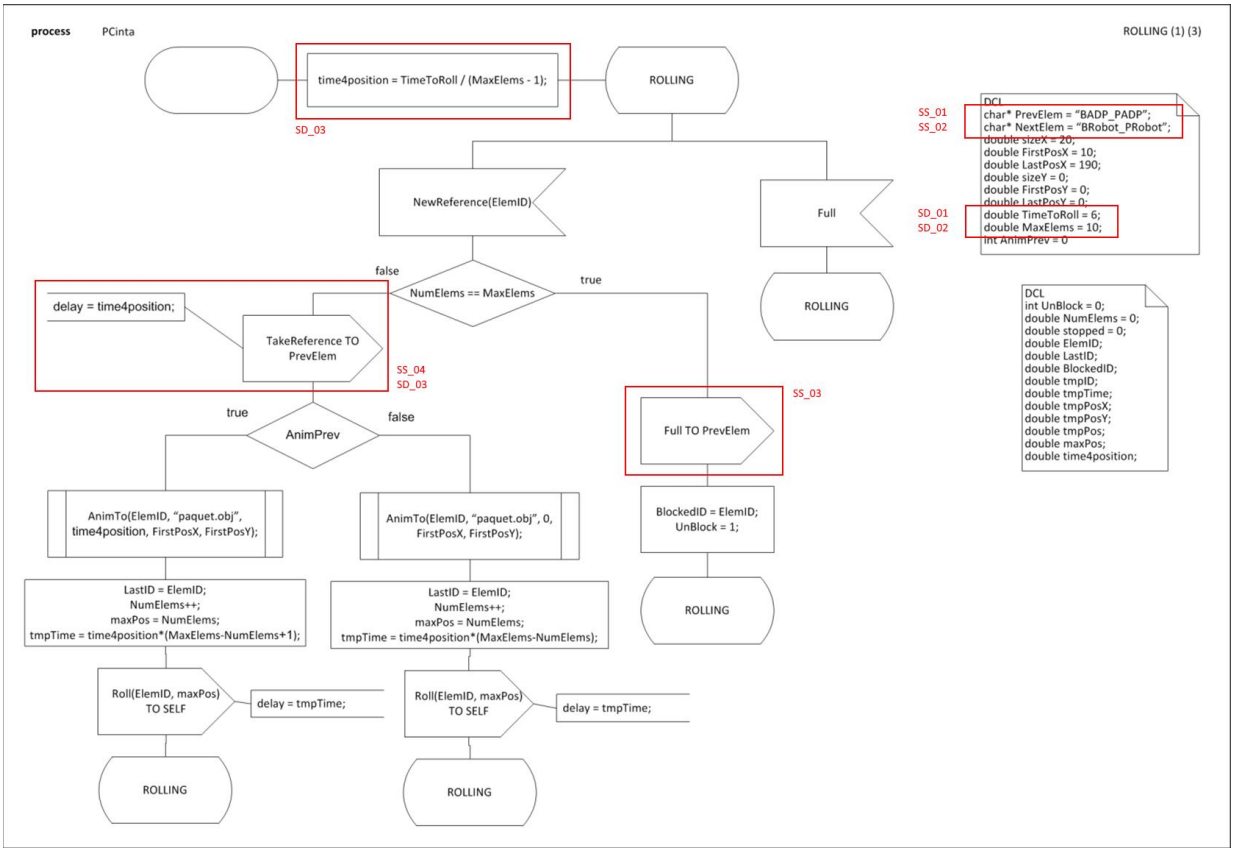

Figure 2. PCinta for the ROLLING state (1/3). Here is presented the PROCESS instantiation. The time4position variable is a Systemic Data assumption that defines the movement of the elements on the conveyor. It states that the time needed to arrive at a specific position depends on the TimeToRoll (defined as 6 s to cross the conveyor) and the number of positions of this conveyor (defined in MaxElems as 10). This Specification and Description Language diagram details unambiguously and completely the assumptions of the model. However, like any conceptual model, it is not going to depict the simplification assumptions used.

Table 5. Assumptions table. The *TimeToRoll* variable is not yet validated on this iteration; hence all the assumptions that are related to it are considered valid but not true. In this iteration, we also detect that the maximum number of elements that can be on a conveyor is not 10; hence SD_02 is false; we must go to the factory and measure the new number of elements that can be on the conveyor.

TYPE	IDENTIFIER	DESCRIPTION	LOCATION	STATUS	REVIEW CONDITION
STRUCTURAL	SS_01	The previous element of the conveyor is the machine named "BADP_PADP".	PCinta	true	If BADP_PADP is modified
STRUCTURAL	SS_02	The next element to the conveyor will be the robot "BRobot_PRobot".	PCinta	true	If BRobot_PRobot is modified
STRUCTURAL	SS_03	When the conveyor is full, it sends a message to the previous element to stop this machine.	PCinta	true	never
STRUCTURAL	SS_04	When the conveyor has empty spaces, the previous element puts a new box in the conveyor and starts its movement to the position that lasts a time defined in SD_03.	PCinta	true	never
DATA	SD_01	The time needed to cross the conveyor is 6 s, defined by the *TimeToRoll* variable.	PCinta	valid	It *TimeToRoll* is modified
DATA	SD_02	The maximum number of elements on the conveyor is 10, defined by the *MaxElems* variable.	PCinta	false	If *MaxElems* is modified
DATA	SD_03	The time needed for the element to reach its position on the conveyor is represented by the expression $\frac{TimeToRoll}{(MaxElems-1)}$.	PCinta	valid	never

The textual representation of the assumption in the conceptual models owns ambiguity, but this document needs to be kept until not all the parts involved in the project feel confident with the formal language used to define the conceptual model that finally replaces this table, except for the simplification assumptions that may not be represented on the conceptual model.

Following the proposed approach, once we have the conceptual model and the assumptions from Table 3, we select a set of tests that covers all the assumptions we want to validate in all the Validation Processes we follow. This allows us to define a table such as Table 6.

Table 6. Validations to be completed in the project.

PHASE	VALIDATION METHOD	SYSTEMIC STRUCTURAL	SYSTEMIC DATA	SIMPLIFICATIONS
CONCEPTUAL MODEL VALIDATION	Face validation	+		+
DATA VALIDATION	Goodness-of-fit		+	
OPERATIONAL VALIDATION	Black Box	+	+	+
EXPERIMENTAL VALIDATION	Sensitivity analysis	+	+	
VERIFICATION	Symbolic debugging	+		
ACCREDITATION	Acceptance testing	+	+	+
SOLUTION VALIDATION	Field testing	+	+	+

The selection of the validations methods in each phase must be made to validate. Before the Accreditation and Solution Validation phases, all the assumptions must consider the resources (time, personnel) we have on the project. In each phase, we apply the test and analyze different assumptions from the table. If some assumptions remain not validated, we can add more tests to complete the validation. In short, we select this subset of validations intending to ensure that all the assumptions are tested by a validation method and that all the phases are also covered. Figure 1 helps to understand the flow we will follow in the iterative modeling process, while the use of the assumptions table and Table 3 help to select

the more appropriate tests. Table 6 is the one we follow in this project. However, other validation approaches can be made to achieve the complete validation of the assumptions and the different phases.

7. Conclusions

The development of a model must be guided by the needed Validation, Verification, and Accreditation processes. This defines an iterative approach until the achievement of the Accepted Solutions, a product that will be implemented in the system. The approach presented in this paper encompasses all the needed actions, products, and validation processes that must be done to achieve a valid model. Moreover, considering the possibility that the model belongs to a Digital Twin, the validation of the model continues while it is in use; this iterative approach systematizes the Solution Validation, allowing understanding and detection of the assumptions that become false due to incorrect understanding or modifications of the analyzed system. Therefore, the prototyping approach, often used in Industry 4.0 to implement different testbeds, is supported by this approach. In a Digital Twin, composed of the Digital Master, which represents the models, and the Digital Shadows, representing the data, obtained from the model and the system, the continuous validation process done in the Solution Validation can cause the reevaluation of the different products obtained through the modeling process.

The definition of a model through its assumptions is the key element that rules its behavior, depicting the suggested causality rules. However, not all the assumptions used in a model have the same effect on its definition. The classification of the assumptions simplifies the model's reevaluation and maintenance. Furthermore, the tests we can use to prove the validity of the model are not working with any typology of assumptions; in our taxonomy, the three classes of assumptions that exist are Systemic Structural assumptions, Systemic Data assumptions, and Simplification assumptions. The use of a taxonomy allows focusing our efforts on a selected subset of the validation tests; the W3H table is the first step to understanding and systematizing the validation test selection. We show that during the iterative process, it is not necessary to ensure that all the assumptions are valid. Sometimes it could be desirable to work with no validated Systemic Data assumptions to validate the Systemic Structural assumptions, or to obtain data related to extreme conditions situations to perform the validations. Working with the proposed approach clarifies the need to increase the Systemic assumptions to represent better and with more detail the structure and the data. On the other hand, we want to decrease the number of Simplification assumptions used to represent a deeper understanding of the system.

The definition of an assumptions table can also be a valuable tool for a company that wants to perform a digitalization of its processes, not only to drive the modeling process. We also note that we must take care of expiration conditions for the assumptions, like the data expiration for the Systemic Data assumptions, to ensure that a model is valid during the life span of the model and take care of any modification in the system that will affect the Systemic (Data and Structural) assumptions. Modifications in the technology, or improvements to the scientific framework used to develop the model, will eventually trigger some simplification assumptions to be discarded.

The life cycle of the modeling process we present depicts all the actions, processes, and products one can obtain in an iterative validation and verification process. This diagram understands the specific features of the new Industry 4.0 requirements and the continuous Validation Verification and Accreditation process that must be done in the context of a Digital Twin creation. The validation and Verification process are the key aspects that must drive any simulation as proposed on the iterative cycle. No model can be considered true but useful, and this depends strongly on how this validation process has been done. The review and the classification of the different Validation techniques one can use will serve precisely to simplify, orientate, and accelerate this Validation process.

This work proposes a complete life cycle guided through the Validation and Verification processes that simplifies the assumptions used and clarifies the products obtained in

each phase. Following the proposed process guides simplifies the Validation Verification and Accreditation actions, increasing the confidence of both modelers and Stakeholders in the final solution.

Finally, we want to mention that, at the beginning of this work, we presented a juxtaposition between the model and the system. Notice that with this approach and understanding the possibility of the interaction between model and system through the Solution Validation, a new entity emerges, including both elements, the system and the Digital Twin.

Funding: This research received no external funding.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A

Appendix A.1. VV Tests

Here we present some validation and verification tests that can be applied to modeling. They are based on [54,55] and contextualized following our proposal. We are not focused on analyzing if the tests are supported by software to facilitate its application, but just mention that more and more software exists that tries to automatize the software creation, like [59,60], and also independent institutions [61,62] focused on the Validation of mission-critical software.

Appendix A.1.1. Informal and Static Validation Techniques

In all the tests of this category, no results will be analyzed since no execution of the model will be done; hence the correctness of the Systemic Data assumptions used on the model cannot be tested here. The assessment of the model will be completed based on the subjective knowledge of the experts by analyzing the documentation, the processes, and the conceptual model.

Conceptual Model Validation

As an example of a test for this category on **Face validation** [11,55], the experts analyze the conceptual model description. From this analysis, they can recognize the correctness of a model. This validation is focused on conceptual model representation, hence on the analysis of the Systemic Structural assumptions that the model conceptual model represents, one can analyze the correctness of the relations, and one can discuss the missing assumptions that do not appear on the conceptual model, or some assumptions that seem oversimplify the model, the Simplification assumptions.

Walkthroughs, **Inspections,** and **Reviews** are focused on detecting documentation faults based on the expertise of a team of experts. Each of these techniques differs in the steps that must be followed and how the teams will be selected. On these types of validations, only Systemic Structural and Simplification assumptions can be analyzed. A similar case is an **Audit** where the simulation study will be audited.

Desk Checking refers to the individual analysis of the work done. Here we are focused on the informal and static analysis of the model, not in any sense the analysis of the results obtained from the model or the correctness of the distributions used (that can be done using statistical techniques). Hence, here we test the Systemic Structural and simplifications assumptions, like **Documentation consistency checking**, which analyzes the completeness of the documentation to ensure that all is up to date.

Traceability Assessment is a technique to ensure that the modeling process accomplishes all the requirements and that the requirements match the design. Currently, there is some effort to systematize and automate these methods exists, like [63,64], but since this process is not fully automated and relies on subjective appreciations, we keep this technique informal. This will be similar to **Interface Analysis**, both for User Interface

analysis [65] and for Model Interface Analysis [66], where some subjective approaches still must be made.

All these techniques allow us to provide insights regarding the correctness of the Systemic Structural and Simplification assumptions but not regarding the Systemic Data assumptions since no statistical tests have been done, and no model execution is done. These types of techniques can be done in the initial stages of the model development, in the Conceptual Model Validation process (but also can be applied in other stages).

Appendix A.1.2. Informal and Dynamic Validation Techniques

In the Informal/Dynamic category, the model will be executed, but the validity of the model will be recognized through the subjective analysis of the experts. No statistical, mathematical, or computational methods will be applied here.

Operational Validation Process

Visualization/Animation tests [55,67] are focused on a graphical, usually a virtual representation, of the simulation model during its executions. We are focused on detecting if the behavior of the elements we see is consistent with the knowledge we have of the system. On **Turing tests** [5,11,68], the simulator generates results that are merged with system data. Again, by examining the documentation that contains simulation results and system data, the experts determine what the reports generated by the simulator are. Statistical analysis can be done, as presented in [68], to detect if the model accuracy is enough. This is a dynamic test, needs the execution of the simulation model, and is an informal test since it is based on expert knowledge. Since the data will be used subjectively by the expert to do the classification, we will review not only the structure of the model but also the correctness of the data we use.

Other Informal/Dynamic tests are **Graphical Comparisons**, which will be based on the analysis of the graphical output of the simulation model and the experts again ensure its validity, or **Object-Flow Testing**, which analyzes the behavior of a specific object (usually an entity) during its execution, and the experts validate its correctness.

In all these tests, we analyze the Systemic assumptions, but we can also discuss the correct use of the Simplification assumptions. Hence these tests encompass all the assumptions that compose the model (as a whole). These tests can be used in the Operational Validation process.

In this category, we can find **Compliance Testing**, also known as Conformance testing. It is a non-functional testing technique that is done to validate the authorization, performance, security, or standards, among other non-functional features, and to detect whether the code of the model accomplishes the organization's prescribed standards: see [69]. Although this type of test refers to non-functional testing, this must be included in the operational Validation process to ensure that the model code meets the needed requirements that must be Accredited on the Acceptance Testing. Systemic Structural assumptions will be tested.

Accreditation Process

Product Testing is the set of tests to be done by the developer before the Acceptance testing. **Acceptance Testing** is not a single test but a set of tests conducted to determine if the requirements of a contract are met. It may involve different tests, some of which can be informal. This test is conducted by the customers and is focused on the model's capabilities as the last step to start using the model in a production environment. Like this, **Alpha** and **Beta Testing** are not referring to a specific technique, but operational testing is done to test the alpha or Beta stage of the product.

These tests are not referring to a specific test but a set of tests that must be done to ensure the correctness of the project. In that sense, they are in the Informal category since the techniques that one can apply can be either formal or informal. If all the tests to be conducted in our accreditation process are formal, then we can move them to the

Formal/Dynamic category. In these tests, we are focused on the validation of all the assumptions.

Appendix A.1.3. Formal and Static Validation Techniques

In the Formal/Static category, we will use formal techniques over the model representation, but as they are static techniques, no execution of the simulation model will be done.

Conceptual Model Validation Process

In **Cause-Effect Graphing**, and specifically on **State Transition analysis**, one will be focused on analyzing how the model change from one state to another depending on the events received in a specific state. We will be focused on the analysis of the conceptual model. As an example, in a Petri Net, we will create the reachability tree [38], or in Specification and Description Language, the reachability graph [70]. **Induction**, **Inference,** and **Logical Deduction** are based on the premise that if a conclusion can be justified based on its premises, this conclusion will be validated, hence true. **Lambda Calculus** works with the model representation, being transformed to a single string (large) to apply correctness proofs, similar to the **Predicate Calculus,** where we will focus on the trueness of a predicate that is formally represented on the conceptual model. **Predicate Transformations** [71] analyze the transformation of the states of the model outputs to the mode inputs. This defines the semantics of the model that can be used to perform the validation process. **Proof of Correctness** depends on the nature of the formal language used to express the model since it needs a mathematical structure to apply the technique.

In these methods, we will be focused on the Systemic Structural assumptions, with a glimpse of the Simplification assumptions.

Verification Process

Control analysis, **Data Analysis** (data dependency; data flow), **Semantic Analysis**, **Structural Analysis**, **Syntax Analysis**, and **Fault/Failure Analysis** are formal techniques that are focused on understanding if the model has been correctly coded rather than describing if the model is correct (that can, however, be also detected here). Hence these techniques will be focused on the Verification process.

Notice that on Verification, we are focused on understanding if we do the model coding correctly, not on analyzing if the model assumptions are correct. For this, verification can provide only insights regarding the Systemic Structural assumptions.

Data Validation Process

One special category is the **Goodness-of-fit** tests that analyze if the distributions we use on the model are accurate in understanding the data we have, see [72]. These kinds of tests usually are related to Systemic Data assumptions. Notice that this is a static technique since here we are only focused on analyzing the distributions we will use in our model, hence is no need to execute the model.

Appendix A.1.4. Formal and Dynamic Validation Techniques

In the Formal/Dynamic category, we can find a plethora of different techniques that will be focused on the validation of Systemic assumptions. However, not all these techniques are going to be focused on the Structural and Data assumptions. Depending on how we use the data from the model, we will be focused on Structural assumptions, Data assumptions, or both.

Operational Validation Process

The **Degenerative** Tests analyze the model's behavior-modifying, the values of input and some selected internal values. The objective is to test if the modification of these

parameters is coherent with the expected result. As an example, if we increase the service time of a server, we expect that the number of elements in the queue will increase. Like this, on **Extreme Condition Tests** is supposed that the model structure and outputs should be credible although using an extreme and unlikely combination of values for the variables. On **Fixed Values** tests, we analyze the outputs for well-known values for the parameters of the model. In this test, we look at the outputs to compare them with the expected results. Like this, **Fault/Failure Insertion Testing** introduces submodules that are not working correctly to detect the incorrect model behavior as expected.

Following other approaches, the **Assertion Checking** test analyzes if, in some parts of the code, an undesired behavior appears through the definition of small pieces of code that are going to be checked. Like this, **Symbolic Execution** uses symbolic values rather than real values to verify the accuracy of the output. **Path Analysis** executes the model with test data to analyze the paths they follow, and going a little further, on **Inductive Assertions,** all input-to-output relations are transformed to assertions and form it is analyzed if on all the paths the assertions remain true. **White-Box Testing** is focused on the internal flows of the model and how it is coded rather than on the results obtained.

In these tests, we are focused on understanding if the relation between the elements is correctly described. Therefore, these tests focus on the Systemic Structural assumptions. Note that in these tests, we analyze if the relations between the model elements are correct. Often no Simplification assumptions are tested here.

Predictive validation (using Historical Data) allows us to understand if the model is behaving as expected, at least for a scenario that is reproducing the behavior of an existing system. In **comparison with other models**, where the underlying idea is that if other models work fine, their outputs (selecting appropriate inputs) must be similar. **Black Box** validation analyzes the model as a Black Box [51,73], hence as a whole, implying that the validation is done over the complete set of assumptions. Like this, **Sub-model/Module Testing**, like **Bottom-Up Testing** and **Top-Down Testing**, divides the model into different submodules and performs an Operational Validation for each submodule. As an example, we can apply a Black Box validation to ensure the correctness of each submodule. This will be similar to **Partition analysis**, which relies on the comparison of different model subparts with the operation we currently know.

Looking at these tests, we see that the model is tested as a whole, for the assumptions tested are all the Systemic and the Simplification assumptions. Here the data used is relevant since we will use the results for the analysis.

Solution Validation

Although a plethora of techniques can be applied to Solution Validation, two tests will be interesting to be used for this category.

The first is **Field Testing**, which implies putting the model in an operational situation, in the situation that it will be used, as an example, as a digital twin of some factory subsystem and see how it operates. This testing must be done formally; although the expert's involvement could be interesting, the implementation we did (on the system) modifies the system. Hence we must be aware of implementing formal mechanisms to ensure the proper validation of the models without relying on the expert's continuous intervention.

Along with this method, to perform the validation, we must ensure that the modification of the model does not introduce new errors. This analysis is named **Regression Testing**, which analyzes the model continuously with the previous datasets used (and passed).

Solution Validation is a key Validation process in the frame of Industry 4.0, where the digital twin must be continuously validated because continuously will be modified because of the continuous modifications due to implementations on the system.

Verification Process

The usual **Debugging** of the code will also be located in this category, where we can use a plethora of techniques, like **Symbolic Debugging**, using breakpoints on the tool used to perform the code to see the variable's values during this coding process. **Execution Testing** refers to a plethora of different techniques that can be applied (monitoring; profiling; tracing), with the main intention of testing being if the execution is performing correctly. It is a technique that is related to the coding of the model conducted during the debugging process. **Interface Testing** is focused on the analysis of the interfaces between the data (input/output mechanisms), the model (between the different sub-models), and the user and the model (being interesting for human-in-the-loop and interactive and training models).

All these techniques are focused on determining if the model has been correctly coded, not on the correctness of the assumptions used. However, during the verification process, we can detect issues in the Structural assumptions of the model.

Data Validation Process

Several **Statistical Techniques** can be applied to the outputs of the model to compare them with the system data to test the overall Systemic assumptions, but often are focused on testing the Systemic Data assumptions.

On the **Event Validity,** we will compare the occurrences of selected events with the real occurrence of those events in the system. As an example, the number of "broken" event occurrences in a specific machine of the model. This kind of test can be useful to test the Systemic Data assumptions because usually, the events that rule the behavior of a simulation model are defined using a known probability distribution or an empirical distribution obtained from a database.

These tests, although they can test the overall assumptions since they are focused on the data analysis, can be used to test Data assumptions using a preliminary or complete version of the model.

Experimental Validation

Analyze the **variability of the parameters** and perform a **sensitivity analysis** that allows analyzing the factors that have the greatest impact on the performance measures. This allows for determining what elements must be modeled carefully and detecting errors in the definition of the relations of the model elements. In this sort of test, we are focused on the Systemic Structural and Data assumptions. We can detect if the probability distributions are correctly represented and if the relations between the different elements are correctly implemented. As an example, if we add between two model elements a causal relation, while in the system only a correlative relation exists, we are introducing an error that can be detected with this test.

Those tests are related to Experimental validation since we need to define an experimental framework and select the variables and factors we want to analyze.

References

1. Bahill, A.T.; Dean, F.F. What is systems engineering? a consensus of senior systems engineers1. *INCOSE Int. Symp.* **1996**, *6*, 500–505. [CrossRef]
2. Balci, O. Golden Rules of Verification, Validation, Testing, and Certification of Modeling and Simulation Applications. *SCS MS Mag.* **2010**, *4*, 1–7. Available online: http://www.springsim.org/magazines/2010-10/index_file/Files/Balci.pdf (accessed on 8 October 2014).
3. Stachowiak, H. *Allgemeine Modelltheorie*; Springer: Vienna, Austria, 1973.
4. Box, G.E.P. Science and Statistics. *J. Am. Stat. Assoc.* **1976**, *71*, 791–799. [CrossRef]
5. Law, A.M. How to build valid and credible simulation models. In Proceedings of the 2009 Winter Simulation Conference, Austin, TX, USA, 13–16 December 2009; pp. 24–33.
6. Carson, J.S. Introduction to modeling and simulation. In Proceedings of the Winter Simulation Conference, Orlando, FL, USA, 4–7 December 2005; pp. 16–23. [CrossRef]
7. Robinson, S. *Simulation: The Practice of Model*; John Wiley & Sons, Ltd.: Hoboken, NJ, USA, 2003.

8. Peck, S.L. Simulation as experiment: A philosophical reassessment for biological modeling. *Trends Ecol. Evol.* **2004**, *19*, 530–534. [CrossRef] [PubMed]
9. Tsioptsias, N.; Tako, A.A.; Robinson, S. Are "wrong" models useful? A qualitative study of discrete event simulation modeller stories. *J. Simul.* **2022**, 1–13. [CrossRef]
10. Sargent, R.G. Verification and Validation of Simulation Models: An Advanced Tutorial. In Proceedings of the 2020 Winter Simulation Conference (WSC), Orlando, FL, USA, 14–18 December 2020; pp. 16–29. [CrossRef]
11. Sargent, R.G. Verification and validation of simulation models. *J. Simul.* **2013**, *7*, 12–24. [CrossRef]
12. Kelton, W.D. Experimental design for simulation. In Proceedings of the 2000 Winter Simulation Conference Proceedings (Cat. No.00CH37165), Orlando, FL, USA, 10–13 December 2000; Volume 1, pp. 32–38. [CrossRef]
13. Robinson, S. Conceptual modelling for simulation: Progress and grand challenges. *J. Simul.* **2019**, *14*, 1–20. [CrossRef]
14. Everett, G.D.; McLeod, R. *Software Testing: Testing Across the Entire Software Development Life Cycle*; Tsinghua University Press: Beijing, China, 2007.
15. Yu, J. Research Process on Software Development Model. *IOP Conf. Ser. Mater. Sci. Eng.* **2018**, *394*, 032045. [CrossRef]
16. Ismail, S.; Dawoud, D.W. Software Development Models for IoT. In Proceedings of the 2022 IEEE 12th Annual Computing and Communication Workshop and Conference, CCWC 2022, Las Vegas, NV, USA, 26–29 January 2022; pp. 524–530. [CrossRef]
17. Balci, O. A life cycle for modeling and simulation. *Simulation* **2012**, *88*, 870–883. [CrossRef]
18. Balci, O. Quality Indicators Throughout the Modeling and Simulation Life Cycle. *Concepts Methodol. Model. Simul. A Tribut. Tuncer Ören* **2015**, 199–215. [CrossRef]
19. Yilmaz, L.; Chakladar, S.; Doud, K. The Goal-Hypothesis-Experiment framework: A generative cognitive domain architecture for simulation experiment management. In Proceedings of the 2016 Winter Simulation Conference (WSC), Washington, DC, USA, 11–14 December 2016; pp. 1001–1012. [CrossRef]
20. Lorig, F.; Lebherz, D.S.; Berndt, J.O.; Timm, I.J. Hypothesis-driven experiment design in computer simulation studies. In Proceedings of the Winter Simulation Conference, Las Vegas Nevada, NV, USA, 3–6 December 2017; pp. 1360–1371. [CrossRef]
21. Lorig, F.; Becker, C.A.; Timm, I.J. Formal Specification of Hypotheses for Assisting Computer Simulation Studies. In *TMS/DEVS Symposium on Theory of Modeling & Simulation (TMS/DEVS 2017)*; Society for Computer Simulation International: San Diego, CA, USA, 2017; Volume 49, pp. 204–215. [CrossRef]
22. Lorig, F. *Hypothesis-Driven Simulation Studies*; Springer Fachmedien Wiesbaden: Wiesbaden, Germany, 2019.
23. Ramos, A.L.; Ferreira, J.V.; Barceló, J. Model-Based Systems Engineering: An Emerging Approach for Modern Systems. *IEEE Trans. Syst. Man Cybern. Part C Appl. Rev.* **2011**, *42*, 101–111. [CrossRef]
24. Lorig, F. Hypothesis-Driven Simulation. In *Hypothesis-Driven Simulation Studies*; Springer Fachmedien Wiesbaden: Wiesbaden, Germany, 2019; pp. 137–170.
25. Rincon, G.; Alvarez, M.; Perez, M.; Hernandez, S. A discrete-event simulation and continuous software evaluation on a systemic quality model: An oil industry case. *Inf. Manag.* **2005**, *42*, 1051–1066. [CrossRef]
26. Fonseca i Casas, A.; Fonseca i Casas, P.; Casanovas, J. Analysis of Applications to Improve the Energy Savings in Residential Buildings Based on Systemic Quality Model. *Sustainability* **2016**, *8*, 1051. [CrossRef]
27. Brade, D. Enhancing modeling and simulation accreditation by structuring verification and validation results. In Proceedings of the 2000 Winter Simulation Conference, Orlando, FL, USA, 10–13 December 2000; pp. 840–848. [CrossRef]
28. Fonseca i Casas, P. Simulation hypotheses. In Proceedings of the SIMUL 2011, Barcelona, Spain, 23–29 October 2011; pp. 114–119.
29. Popper, K. *The Logic of Scientific Discovery*; Routledge: London, UK, 1959.
30. Epstein, J. Why Model? *J. Artif. Soc. Soc. Simul.* **2008**, *11*, 12. Available online: http://jasss.soc.surrey.ac.uk/11/4/12.html (accessed on 28 January 2023).
31. Teng, P.S. Validation of computer models of plant disease epidemics: A review of philosophy and methodology/Zuverlässigkeit von Computermodellen für Epidemien von Pflanzenkrankheiten: Ein Überblick über Grundgedanken und Methodik," Zeitschrift für Pflanzenkrankheiten und Pflanzenschutz. *J. Plant Dis. Prot.* **1981**, *88*, 49–63. Available online: http://www.jstor.org/stable/43214777 (accessed on 3 February 2023).
32. Naylor, T.H.; Finger, J.M. Verification of Computer Simulation Models. *Manag. Sci.* **1967**, *14*, B92–B106. [CrossRef]
33. Sherratt, E.; Ober, I.; Gaudin, E.; Casas, P.F.; Kristoffersen, F. SDL—The IoT Language. In *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*; Springer: Berlin/Heidelberg, Germany, 2015; Volume 9369, pp. 27–41.
34. Fonseca i Casas, P. *Specification and Description Language for Discrete Simulation*; IGI Global: Hershey, PA, USA, 2013.
35. ITU-T. Specification and Description Language—Overview of SDL-2010. ITU-T Recommendation Z.100. 2019. Available online: http://handle.itu.int/11.1002/1000/14048 (accessed on 3 February 2023).
36. Van der Aalst, W.M.P. Timed Coloured Petri Nets and Their Application to Logistics. Ph.D. Thesis, Eindhoven University of Technology, Eindhoven, The Netherlands, 1992. [CrossRef]
37. Guasch, A.; Figueras, J.; Casanovas, J. Conceptual modeling using Petri Nets. In *Formal Languages for Computer Simulation: Transdisciplinary Models and Applications*; IGI Global: Hershey, PA, USA, 2013.
38. Wang, J. Petri Nets for Dynamic Event-Driven System Modeling. In *Handbook of Dynamic System Modeling*; Fishwick, P.A., Ed.; Chapman & Hall: Gainesville, FL, USA, 2007; pp. 1–2, 7–9.

39. Mutarraf, U.; Barkaoui, K.; Li, Z.; Wu, N.; Qu, T. Transformation of Business Process Model and Notation models onto Petri nets and their analysis. *Adv. Mech. Eng.* **2018**, *10*, 1–21. [CrossRef]
40. Zeigler, B.P.; Muzy, A.; Kofman, E. *Theory of Modeling and Simulation*; Elsevier: Amsterdam, The Netherlands, 2019. [CrossRef]
41. Concepcion, A.; Zeigler, B. DEVS formalism: A framework for hierarchical model development. *IEEE Trans. Softw. Eng.* **1988**, *14*, 228–241. [CrossRef]
42. Wainer, G.; Giambiasi, N. Timed Cell-DEVS: Modeling and Simulation of Cell Spaces. 2001. Available online: http://link.springer.com/chapter/10.1007/978-1-4757-3554-3_10 (accessed on 20 November 2013).
43. Fonseca i Casas, P. Transforming classic Discrete Event System Specification models to Specification and Description Language. *Simulation* **2015**, *91*, 249–264. [CrossRef]
44. Boukelkoul, S.; Redjimi, M. Mapping between Petri nets and DEVS models. In Proceedings of the 2013 3rd International Conference on Information Technology and e-Services (ICITeS), Sousse, Tunisia, 24–26 March 2013; pp. 1–6. [CrossRef]
45. Vangheluwe, H.L. DEVS as a common denominator for multi-formalism hybrid systems modelling. In Proceedings of the CACSD. Conference Proceedings. IEEE International Symposium on Computer-Aided Control System Design (Cat. No.00TH8537), Anchorage, AK, USA, 25–27 September 2000; pp. 129–134. [CrossRef]
46. Fonseca i Casas, P. Co-simulation Using Specification and Description Language. In Proceedings of the 2013 Winter Simulation Conference: Simulation: Making Decisions in a Complex World, Washington, DC, USA, 8–11 December 2013; pp. 4022–4023. Available online: http://sdlps.upc.edu (accessed on 3 April 2022).
47. Garrido-Soriano, N.; Godoy, A.; Pujols, W.-C.; Garcia, J. Solution Validation for a Double Façade Prototype. *Energies* **2017**, *10*, 2013. [CrossRef]
48. Brozovsky, J.; Haase, M.; Lolli, N. Validation of a Digital Twin with Measurement Data. In Proceedings of the 39th AIVC—7th Tight Vent & 5th Venticool Conference, Juan-les-Pins, France, 18–19 September 2018; pp. 1–10.
49. Sargent, R.G. An interval statistical procedure for use in validation of simulation models. *J. Simul.* **2015**, *9*, 232–237. [CrossRef]
50. Balci, O. Verification, validation, and certification of modeling and simulation applications. In Proceedings of the 2003 Winter Simulation Conference, New Orleans, LA, USA, 7–10 December 2003; pp. 150–158.
51. Robinson, S.; Brooks, R.J. Independent Verification and Validation of an Industrial Simulation Model. *Simulation* **2009**, *86*, 405–416. [CrossRef]
52. Chew, J.; Sullivan, C. Verification, validation, and accreditation in the life cycle of models and simulations. In Proceedings of the 2000 Winter Simulation Conference, Orlando, FL, USA, 10–13 December 2000; pp. 813–818. Available online: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=899879 (accessed on 8 October 2014).
53. Kari, J. Theory of cellular automata: A survey. *Theor. Comput. Sci.* **2005**, *334*, 3–33. [CrossRef]
54. Back, G.; Love, G.; Falk, J. The Doing of Model Verification and Validation: Balancing Cost and Theory. In Proceedings of the 2000 System Dynamics Conference, Bergen, Norway, 6–10 August 2000; Volume 7055, p. 31.
55. Balci, O. Verification, validation and accreditation of simulation models. In Proceedings of the Winter Simulation Conference, Atlanta, GA, USA, 7–10 December 1997; p. 3.
56. Page, E.H.; Canova, B.S.; Tufarolo, J.A. A case study of verification, validation, and accreditation for advanced distributed simulation. *ACM Trans. Model. Comput. Simul.* **1997**, *7*, 393–424. [CrossRef]
57. Kim, B.S.; Kang, B.G.; Choi, S.H.; Kim, T.G. Data modeling versus simulation modeling in the big data era: Case study of a greenhouse control system. *Simulation* **2017**, *93*, 579–594. [CrossRef]
58. Casas, P.F.; Palomés, X.P.; Garcia, J.C.; Jové, J. Definition of virtual reality simulation models using specification and description language diagrams. In *SDL 2013: Model Driven Dependability Engineering*; Springer: Berlin/Heidelberg, Germany, 2013; pp. 258–274.
59. Labs, A. Auto Validate. 2022. Available online: https://www.auroralabs.com/product/auto-validate/ (accessed on 3 November 2022).
60. Driscoll, T. Automatic Validation of Software. MATLAB Central File Exchange. 2022. Available online: https://es.mathworks.com/matlabcentral/fileexchange/20037-automatic-validation-of-software (accessed on 3 November 2022).
61. University of Luxembourg. Software Verification and Validation Lab. 2022. Available online: https://wwwen.uni.lu/snt/research/software_verification_and_validation_lab (accessed on 3 November 2022).
62. NASA. Katherine Johnson IV&V Facility. 2021. Available online: https://www.nasa.gov/centers/ivv/home/index.html (accessed on 3 November 2022).
63. Biró, M.; Klespitz, J.; Gmeiner, J.; Illibauer, C.; Kovács, L. *Towards Automated Traceability Assessment through Augmented Lifecycle Space*; Springer International Publishing: Cham, Switzerland, 2016; Volume 633.
64. Rempel, P.; Mäder, P. Continuous assessment of software traceability. In Proceedings of the International Conference on Software Engineering, Rome, Italy, 21–25 August 2016; pp. 747–748. [CrossRef]
65. Rafael, S.; Almeida, V.M.; Neves, M. A Human-Computer Interaction Framework for Interface Analysis and Design. In *Advances in Ergonomics in Design, Proceedings of the AHFE 2019 International Conference on Ergonomics in Design, Washington DC, USA, 24–28 July 2019*; Springer International Publishing: Berlin/Heidelberg, Germany, 2019; pp. 359–369. [CrossRef]
66. Aicher, T.; Schütz, D.; Spindler, M.; Liu, S.; Günthner, W.; Vogel-Heuser, B. Automatic analysis and adaption of the interface of automated material flow systems to improve backwards compatibility. *IFAC-PapersOnLine* **2017**, *50*, 1217–1224. [CrossRef]

67. Balci, O. Validation, verification, and testing techniques throughout the life cycle of a simulation study. *Ann. Oper. Res.* **1994**, *53*, 121–173. [CrossRef]
68. Schruben, L.W. Establishing the credibility of simulations. *Simulation* **1980**, *34*, 101–105. [CrossRef]
69. Softwaretestinghelp. What Is Compliance Testing (Conformance Testing)? Software Testing Help. 2020. Available online: https://www.softwaretestinghelp.com/what-is-conformance-testing/ (accessed on 14 June 2020).
70. Bourhfir, C.; Aboulhamid, E.; Dssouli, R.; Rico, N. *Test Case Generation Approach for Conformance Testing of SDL Systems*; Woodhead Publishing Limited: Cambridge, UK, 2001; Volume 24.
71. Garoche, P.L. *Formal Verification of Control System Software*; Princeton University Press: Princeton, NJ, USA, 2016.
72. Law, A.M.; Kelton, W.D. *Simulation Modeling and Analysis*; Mc Graw Hill Education: New York, NY, USA, 1991; Volume 2.
73. Kleijnen, J.P. Verification and validation of simulation models. *Eur. J. Oper. Res.* **1995**, *82*, 145–162. [CrossRef]