VISUALIZATION OF VERIFICATION INFORMATION THROUGHOUT THE SYSTEM LIFECYCLE USING A MODEL-BASED TOOL

by

Ashley Thrower

A Thesis Submitted to the Faculty of the

DEPARTMENT OF SYSTEMS AND INDUSTRIAL ENGINEERING

In Partial Fulfillment of the Requirements

For the Degree of

MASTER OF SCIENCE in SYSTEMS ENGINEERING

In the Graduate College

THE UNIVERSITY OF ARIZONA

2022

THE UNIVERSITY OF ARIZONA
GRADUATE COLLEGE

As members of the Master's Committee, we certify that we have read the thesis prepared by: Ashley Thrower
titled: *Visualization of Verification Information Throughout the System Lifecycle using a Model-Based Tool*

and recommend that it be accepted as fulfilling the thesis requirement for the Master's Degree.

Alejandro Salado — Date: Dec 4, 2022

Ricardo Valerdi — Date: Dec 4, 2022

Robert Lepore — Date: Dec 4, 2022

Final approval and acceptance of this thesis is contingent upon the candidate's submission of the final copies of the thesis to the Graduate College.

I hereby certify that I have read this thesis prepared under my direction and recommend that it be accepted as fulfilling the Master's requirement.

Alejandro Salado — Date: Dec 4, 2022
Thesis Committee Chair
Department of Systems and Industrial Engineering

ACKNOWLEDGEMENTS

Dr. Chet Preston: you have my gratitude beyond words for seeing this through with me by guiding, listening, and willingly learning about a new subject matter outside your expertise.

My grandmother, for leaving me a means to complete this. It's beautiful to be reminded of your support for education and for me to take part in its extension.

C'est la vie.

LAND ACKNOWLEDGEMENT

We respectfully acknowledge the University of Arizona is on the land and territories of Indigenous peoples. Today, Arizona is home to 22 federally recognized tribes, with Tucson being home to the O'odham and the Yaqui. Committed to diversity and inclusion, the University strives to build sustainable relationships with sovereign Native Nations and Indigenous communities through education offerings, partnerships, and community service.

Table of Contents

Table of Figures

Abstract

The continual increase of system complexity in industry has demanded more innovative ways to handle its organization and development. One organizational component that can be easily mishandled is that of verification. Without streamlined or highly visible verification and validation, miscommunication can arise between the involved parties such as the client, the systems engineer, the test engineers, and the other engineering groups involved in the design and fabrication of the product. This thesis leverages the shift towards model-based environments to produce a verification framework to be implemented within the Model-Based Systems Engineering field. The framework is implemented in a modeling tool to produce an information view capturing detailed verification information from element coupling. Using object-oriented design pattern considerations, this approach supports the evolution towards model-centric environments in industry and adaptation for differing verification strategies. It encourages early implementation of verification in the system development lifecycle and can be the starting point for a standard model-based verification information output. This structure enhances visibility of information across all involved parties and allows for growth when integrated with a system architecture model and additional digital environment artifacts. As a foundation, this visibility enhanced representation may be combined with other verification strategies to enable automation and/or optimization of a system's design.

1. Introduction

Model-Based Systems Engineering (MBSE) creates opportunities for new ways of digitally representing complex systems and new opportunities for system and development effort organization and project management processes [1,2]. These benefits include support of automation, early verification and validation, strengthened testing, better accessibility of information, increased traceability, multiple viewpoints/perspectives, reusability, and a higher level of support for integration, among other value areas to be quantified [2,3].

MBSE is founded on informational models, or the relation of the system's elements and information to each other [4,5]. The implementation methods align with organizational digitization, an initiative championed by the International Council of Systems Engineering (INCOSE) [6]. Described by INCOSE, it is "the formalized application of modeling to support system requirements, design, analysis, verification and validation activities beginning in the conceptual design phase and continuing throughout development and later life cycle phases" [7]. This leads to a major function of MBSE, the organization and visualization of information for system stakeholders [8].

Before MBSE, the core objectives of a system's engineer were accomplished in a document-centric fashion. This means systems engineering products are mainly produced as documents and information is verbally flowed between the systems engineer and engineers of other specialty disciplines. Product management databases have improved document centralization and configuration management, but it can still be hard to search through and maintain its current information. When trying to reuse this information or find the initial

investigation trail the information came from, time spent on nonvalue-added work increases. The hand-off of information between parties also takes time and leaves opportunity for miscommunication of information across multiple documents [9]. The limitations of this compared to MBSE are the need to manually update and maintain documents constantly, and continuously trace who and what is being affected by changes in the process. Due to this heavy reliance on numerous physical documents, information management in a document-centric environment can allow for inconsistencies, misplacement of information, and difficulties with accessibility [10]. Purely model-centric environments, on the other hand, can reduce time in the process by having interfaces which handle the flow of information to users. Having common formats and a singular environment for information communication can reduce time spent on information gathering. A model-centric environment may also keep its objects in its own database for reuse, eliminating the need to find the information and allowing updates to dynamically affect all associated elements. Modeling tools themselves provide the platform which can support access to various data types. If the organization does not produce its artifacts and activities in a digital environment, the benefits of digital transformation will be lost [2]. Having a central source of truth is one of the primary MBSE goals, enhancing communication, sharing, and management of data, information, and knowledge [3]. Focusing on these aspects in the MBSE realm allows tools to become the interface for the management of systems engineering information.

Due to these benefits, shifts toward MBSE and model-centric environments have occurred in various sectors of industry and are continuing to gain popularity [6,11,12]. The digital transformation relies on adoption, usability, productivity, and new value being provided [3]. However, regardless of its ongoing adoption, there is still room for improvement in the field and quantifying the delivered benefits [13].

One particular facet of system development that may not have seen proportional development in the transition to MBSE. is that of verification [14]. Verification (and validation) of requirements is given the most focus in the latter phases of a project's lifecycle. However, if requirement tracking and incremental verification is included in a model-based application, it would naturally integrate and grow along with the entire system lifecycle. Nevertheless, standard formats for communicating verification information in a model-based tool, like that of a Verification Cross-Reference Matrix (VCRM), are not widespread if they exist [15]. Therefore, an emphasis on project information visibility and veracity is vital to the verification and validation of its associated requirements in the digital domain.

This thesis proposes, an early implementable verification architecture in a model-based environment that allows for information visualization. This structure can incorporate verification planning elements, follow program progression towards verification, promote information sharing and data flows, maintain archiving of artifacts, and allows for the forward progression towards model-centric engineering environments. Producing this intermediary solution could allow stakeholders to consume verification information and open a path for adoption of more complex solutions; for example, increasing efficiency and utilizing optimization techniques in

the verification domain. This would accomplish an increase in quality in MBSE verification, delivery of model information in way that increases stakeholder communication, and familiarity with tool utilization for use of the system information; mentioned as current obstacles in industry [3,16].

Verification information visibility is achieved by creating the verification elements in a modeling tool, coupling the model elements, and producing an information table for verification tracking throughout the system lifecycle to support system verification for stakeholders in industry today. An example of direct application is provided in this work in order to display the interrelations of project/product components utilizing this dynamic, reusable verification framework.

2. Review of the State of the Art

System verification is generally concerned with demonstrating that requirements have been met and showing configuration management of the designed system [17]. The early design stages are vitally important for the correct capture of technical and lifecycle requirements arising from understanding and interpreting stakeholder needs. Early and continual testing provides confidence towards meeting these requirements. The way in which this is implemented in practice is transitioning from traditional, document-based approaches to the virtual or digital domain. Yet, Estefan identified that MBSE efforts have largely focused on the system's architecture, behaviors, and requirements, but largely excludes the verification domain [18]. This is prevalent in the MBSE literature.

Friedenthal, Moore, and Steiner mention the importance of the theory of verification and the elements needed to complete this aspect of systems engineering. However, their work stops after mentioning test case refinement and traceability to the requirements, without providing constructs to model a verification plan. Stakeholders are mentioned frequently, but no bridge for getting the information to the stakeholders, aside from the dictation that it should be populated in the model, is given. The needs of the multidisciplinary engineering teams are also highlighted, listing what informational areas each team has an interest in and responsibility towards, but again the implementation aside from considering its addition to the model is not mentioned. The latter part of the book suggests the SysML model be integrated with other tools to complete system development. This would use the SysML provided test case as an input to an external verification tool, which in turn provides results back to the model. The 2017 request for proposal of a SysML 2.0 showed extension of the modeling concepts to include verification [16,19].

The INCOSE established a working group for the Object-Oriented Systems Engineering Method (OOSEM) in 2000 which utilizes object-oriented concepts to model traditional systems engineering methods, including the extension towards system verification. OOSEM includes stakeholder approaches to analyze the needs and define the logical architecture for design solution space. This structure supports using the system model for verification planning and documentation outputs, as well as executing early verification from the modeled system

behaviors. Their recommendation is for more mature testing points towards integrating the model with external analysis models. The verification domain is suggested to be decomposed as its own architecture to capture the unique resources and components.

A first attempt to use MBSE to model system integration and test is found in Montgomery's *Model-Based Systems Integration: An Instructional Approach* in 2012 [20]. His vision is for evaluating the architecture and design without needing additional products for this purpose, possibly reducing cost and scope of its implementation. This work was followed by Salado who sets the stage for using MBSE for system integration and verification activities at the end of the development cycle [5].

Salado elucidates that the historical strength in MBSE has been in showing early validation of solution concepts and where requirements are to be met, proposing an extension towards the planning domain for system integration and test activities. In this discussion the missing elements pointed out include verification requirements, integration and verification flows, test instrumentation, test plans and predictions, integration and test procedures, and personnel allocation [21–23]. Instead, they are all managed through traditional tools which are not model-based. He points out that this produces risk when planning/conducting system verification. Salado presents the application of an informational model in a real-life setting to couple these elements in a visual process for the verification activities allows for organization, inclusion, and a strategy to determine the optimal system verification path and order.

Salado's work, originally implemented in STRATA (another MBSE language), was transferred to SysML by Selvy et al. to support the development of the Large Synoptic Survey Telescope (LSST). Their approach included creation of a new stereotype for the requirement to add a verification owner, verification methods, verification level, verification requirement, and success criteria as tagged values with enumerated lists for its values. The SysML Test Cases are used to represent the verification events and the verification plan is mapped to them. The requirements are refined by these verification planning elements and the verification activities verify the verification events or test cases. The verification event's activity sequence is dictated using a sequence diagram to show order and constraints. LSST integrates this set up with their planning tools using Primavera software to further allocate the MBSE elements with the project's Integrated Master Schedule (IMS) and Earned Value Management (EVM) system [24]. The conclusion of the LSST project states that the current SysML language covers the design and development phases and excludes the same robustness required for verification and validation planning. The aforementioned stereotype includes some additional fields to bridge this gap to extend the traceability to track verification activity completion [24].

No other paper in this direction is found in the literature until Kraft, with the evolution from MBSE to DE, and the Department of Defense (DoD) strategic intent to transform its operations to leverage digital artifacts [25]. The work is similar to that of Montgomery in that it presents the potential of using digital artifacts to support test and evaluation planning, although he does not present actual application of the idea as Salado and Selvy et al. did [5,20,24]. The

DoD strategy mentions the DoD Architecture Framework (DoDAF) as the methodology for MBSE implementation [25]. DoDAF is limited because it captures the system architecture and information flows. However, this does not support implementation of a verification framework or supporting development/deployment/testing activities integrated with MBSE. To effectively test systems, the strategy focuses on "right data collection infrastructure", utilizing storage solutions, visualization tools, and a virtual operations center which could be integrated with MBSE to achieve [25].

As can be seen, the use of MBSE and/or DE to support verification planning is very scarce particularly when compared with the broad adoption of MBSE and/or DE to support other SE activities. This might be caused by the high cost of incorporating MBSE and the perceived reduction of continued value in MBSE through the system lifecycle [13,16].

There is a need to continue use and evolution of the accepted systems engineering principles as the methods, tools, and uses change. The development of MBSE architecture models and the lengthy studies done on modeling-tool efficiencies identifies an opportunity to close the System Development Life Cycle (SDLC) loop in this environment. Support for this extended practice of SE will continue once the utility of the MBSE tool supports system management and communication with stakeholders, particularly in the verification domain. To employ further use of these methodologies, an emphasis on verification could effectively provide confidence in meeting system requirements during the development of the system and enhance visibility and communication between the distinct parties involved. That is, without needing a fully developed or detailed architecture model, a verification structure can be overlayed and developed concurrently with the design to deliver these benefits. The remainder of this thesis provides a framework to utilize along the path to achieve this goal.

3. Fundamental Concepts

A few basic concepts require further elucidation for adequately inspecting the role of verification throughout the product development process. These concepts are the foundation for the information in the flow of the model verification framework described in Section 4, to meet the objectives identified in the Introduction.

3.1 Systems Engineering Process and Variables

The system lifecycle is a well-known concept in many engineering disciplines, used to capture the phases of product development [26]. This lifecycle divides system development into phases from its inception through the system's retirement. One example of this process is the System Development Life Cycle shown in Figure 1. This particular lifecycle is broken down into the Requirements Analysis phase, Design phase, Implementation phase, Testing phase, and Evolution phase. Product development follows this process by getting requirements from the customer, developing a design, implementing the design with the physical product, testing the product built, and evolving the system as necessary. This cycle is repeated until development and delivery is complete for all aspects of the system.

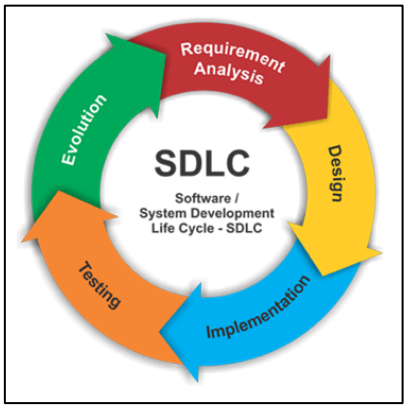

Figure 1: System Development Life Cycle depiction from Hassan Gomaa 2011

The Systems Engineering 'V' in Figure 2 is commonly used to show the systems engineering approach to the system's requirement decomposition and component integration, capturing where verification aligns during the creation process [27,28]. The relationship of system design (left) to system implementation (right) is proven during a program through the matching verification and validation arrows across the 'V'.

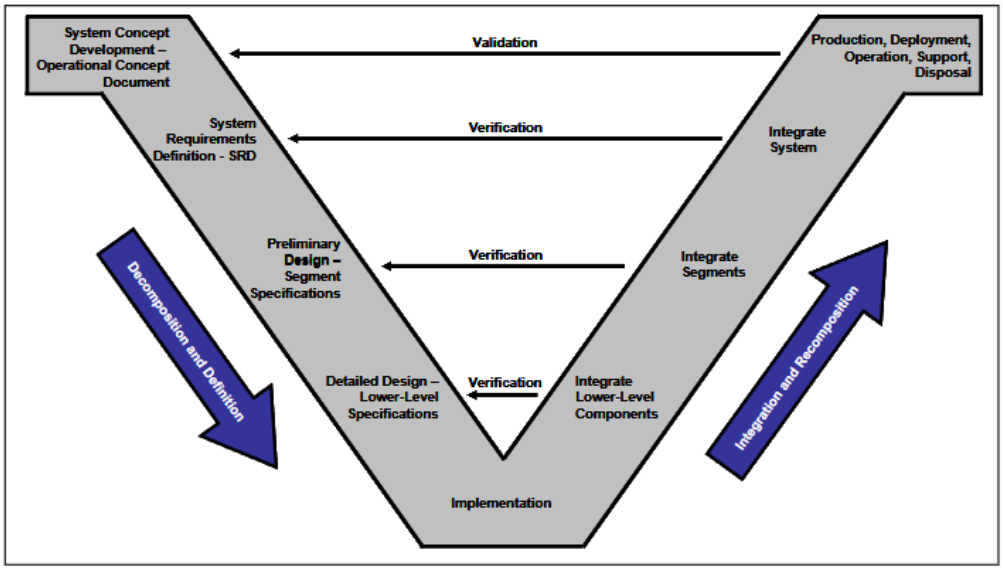

Figure 2: Systems Engineering Vee as depicted by Scukanec and van Gaasbeek 2010, based on Forsberg, Mooz, and Cotterman 2005

From these concepts I have synthesized a flow diagram depicted in Figure 3 for a linear perspective of the major elements I believe are applicable to system verification, maturing with information creation and movement throughout the system lifecycle.

The strategy of how verification is approached and how a particular program handles steps of the verification process can be unique. The inputs and outputs of the system engineering process, however, remain constant allowing them to be universally described by a single flow diagram. By leveraging this unchanging backbone, we can formulate a generalized verification approach for various system structures and programs. The three key steps underlining this foundation are: 1) understanding requirements, 2) capturing system configuration, and 3) recording system performance.

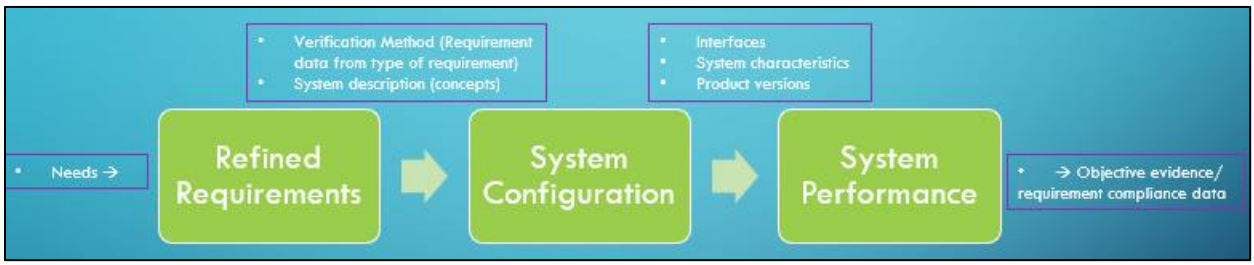

Figure 3: System Verification Element Flow Diagram showing input and output variable for each major aspect of the system lifecycle.

The customer provides system requirements based on their needs as the input, whether directly or through translation from what they express as the product need. First, requirements are assigned verification methods to help determine how the program will prove the requirements have been met. The system architecture, whether conceptual or realized based on an existing system, is developed from the needs and requirements to understand the components of the product being developed and describing the system.

The system configuration(s) captures the product versions, interfaces for integration, and the actual characteristics of the system created. As the system matures, these variables will be fully known and may include prototypes and configurations purely for testing purposes. The requirements may identify some of the system characteristics or they are derived from the system's architecture or component choice, essential aspects of determining when requirements are met.

Lastly, system performance captures the system's data outputs and results from testing. The products here will feed directly into the verification and validation of the project. This testing may be used to further characterize the system, provide testing context, and capture test results. By comparing the results to the requirements and showing product fulfillment by producing evidence of meeting requirements, verification and validation of the system is proved.

3.2 Use of Object-Oriented Design Patterns

In order to take an object-oriented approach towards the problem space, I am using a technique from Alan Shalloway and James Trott's *Design Patterns Explained: A New Perspective on Object-Oriented Design*. The goal of Shalloway's Abstract Factory Pattern (AFP)

is to coordinate families of objects to hand to the system, in our case the program or contractor, during certain situations [29]. The biggest focus of this method is to ensure object variations can be handled through the methods used in the class or, in the case of this thesis, the verification structure presented later in Figure 5.

The AFP is described as a function with inputs, processes, and outputs focused on interface handling for efficiency and response to variation. I have made a representation of Shalloway's depiction in Figure 4. The input is the client's message calling for an output object. In the system development realm, this would be the customer asking a contractor to produce something.

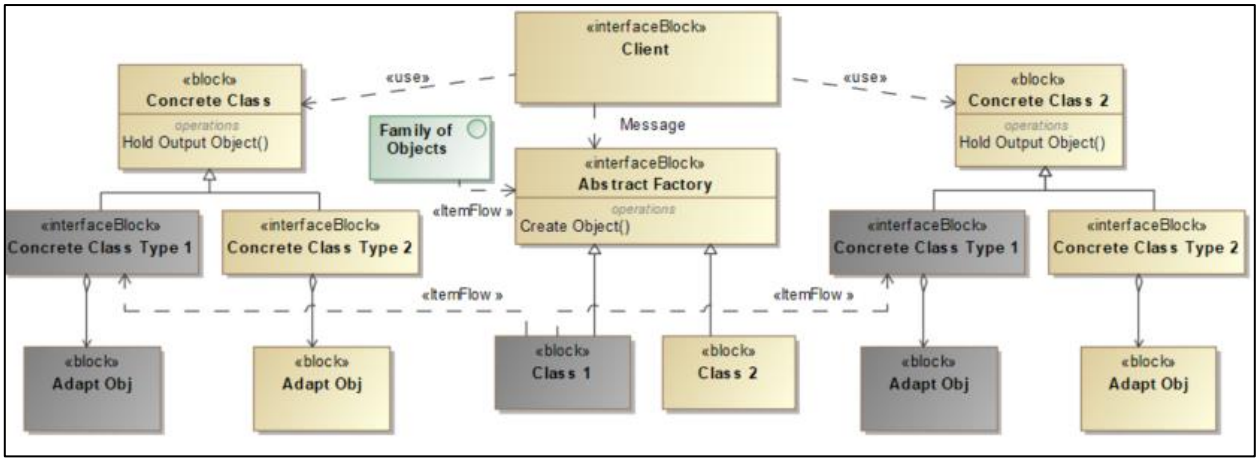

Figure 4: Abstract Factory Pattern Representation

The client knows how to ask for the product and what they will do with it [29]. The receipt of the request goes into the process box referred to as the abstract factory. The purpose of the abstract factory is to be the interface between the request and object creation. This method of using an interface provides separation between the request message and the logic used to create the output object; essentially allowing the developing party (entities part of the abstract factory) to choose the method of implementation freely. The abstract factory uses a method to instantiate a driver to get an expected object. The entities in the factory know the methods to interpret the request and the set the solution is found in. Keeping it abstract allows for the identification of the classes in certain domains while allowing for evolution of methods and expansion of results. In a specific realm, these become *concrete factory* instantiations. Concrete factories use classes that categorize what information is being looked for to get to the output object.

The AFP produces an object to request the output and that output is provided back to the customer in a known format for them to handle. This is done by determining which class will get the object. The classes are created to handle every permutation of the input information so the exact requesting object has the necessary information to go with it to get to the expected result. Adding an additional interface here creates opportunity for adaptation of the requested object to get the desired result. This would be seen when the information is from an outside source or contains variation itself. For example, the request may need to look in a configuration file to determine the path the information will continue down to get the particular result. Other forms

may be stored in external databases or in information elements already saved or pre-existing [29].

When these methods and classes are determined, the process logic is part of the communication shared with the customer to continue development visibility and is part of the engineering documentation. Knowing how this information is handled by the system instantiating the system of interest and the formats required provides improvement in information handoff.

The expected inputs to the abstract factory are referred to as the family of objects. In a large system comprised of many layers of abstract factories, the lower-level factories may be given the family of objects by the main system [29]. The main object groups that systems engineers handle are customer requests, product external interfaces, and system requirements.

Production of the system and its associated information along with system verification is the output to the customer from the pattern as applied to entire system development. Once the framework has been established in this manner, the information flow can be followed for visibility and expected for the duration of product development. Continuous updates for the system developers are available and current status of the design can be reported. When done in a modeling environment, this would not require additional documentation management or creation.

4. Method- Verification Factory Pattern

By adapting the AFP to the verification domain or subject matter, one can show verification as a class or categorical decomposition of the systems engineering domain. As industry progress in the implementation of model-centric environments, the idea would be to integrate this framework as a layer on other models of the system of interest. For example, the verification objects would be directly associated as properties of the system architecture with the remainder of the program's areas of interest such as risk associated, object quantities, schedule, and engineering drawing artifacts.

To make the implementation inclusive, I use a holistic view of system verification in the system engineering domain so it can be used during all phases of the system lifecycle. This demonstrates the ability for information gain over time after implementing the framework, encouraging early implementation of system verification aspects. Information is naturally gained as the system is more completely understood during the design and implementation, giving higher fidelity of data associated with the model and system elements as it develops.

Figure 5 depicts the AFP as applied to system verification. In this image you can see the customer needs being supplied to the systems engineering team carrying out the system verification. The input objects they use are the variables derived in Section 3.1, and the verification activity is provided with the necessary information to carry out the event. The output from the verification activity provides some type of evidence of verification for the requirements and ultimately leads to system validation. This information is used by the customer to see the system was verified and for other results of its performance.

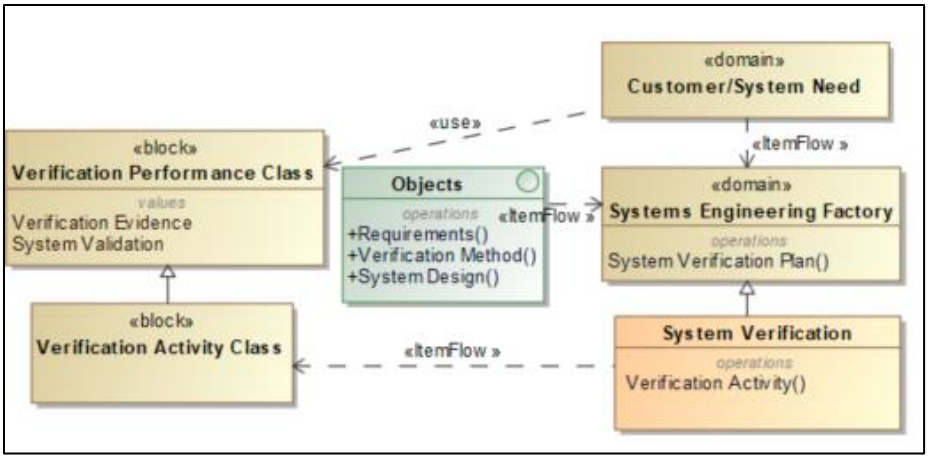

Figure 5: Verification Factory Pattern; adaptation of the Abstract Factory Pattern as applied to the system verification process

The client in the AFP presented is represented by the customer or need for the system. The systems engineers use the family of objects to create the detailed elements of the verification plan (the objects created internally in the factory). The parts of the family of objects are the system requirements, the verification methods, and the system design (objects, components, or behaviors). From the verification standpoint, the driver is the verification method for the creation of specific verification activity classes. The image depicts this as a general class but would contain each event that needs to be performed. This allows unique objects to be created containing the details for a single verification event.

The systems engineering factory uses methods unique to the contractor or program for handling complete resource needs for the event and verification performance determination. The output from the coupling of system information provides the necessary data needed in the verification process whether it be for the systems engineers, developers and testers, or directly for the customer. This ensures the event will be handled correctly.

The information output is flowed to the verification performance class. This is an aggregation of the object information related to the outcome of the verification performance, the procedures that took place, and documentation for the system's final validation.

4.1 Model Coupling

To capture the verification information previously described in a single view and capture the changes and gain of information over time, these elements must have relationships in the modeling tool used. I will use Dassault Systèmes' Cameo Systems Modeler tool as the model-based environment to show the coupling between the major components. These relationships produce the framework to show system verification information and aggregated for the table output.

To enhance communication with the customer, the first area of coupling I am presenting is the relationships between the system concept (what is asked) and the system design proposed. The system concept is the vague architecture created to visually describe the system needs from the customer in an abstract way. The proposed system design is the system architecture using the names the design team will refer to the system as and the name of the hardware being used in order to differentiate between configurations and design ideas. In Cameo, these conceptual blocks are allocated to the actual system blocks representing the design or hardware as shown in Figure 6. One way to think of it is as a product maturity matrix**,** showing the relationships between the product expressed by the customer and the components satisfying these areas as the system is realized. These areas may evolve as the design and components become known but allows traceability to the original system concept.

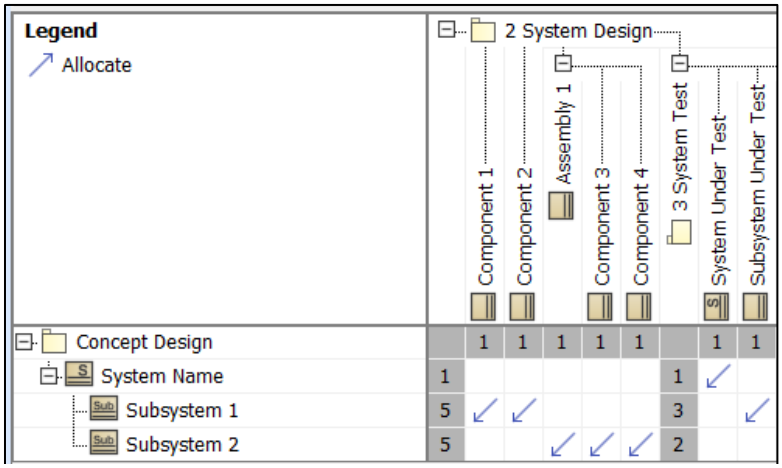

Figure 6: Conceptual System Traced to System Physical Components

The next set of relationships to build keeps the realized components of the system as columns but adds the requirements as rows for requirement tracing through the system, as shown in Figure 7. One would determine all the system components that the requirement encompasses and create relationships to it. This provides the essential parts required for the verification event in which the requirement is tested.

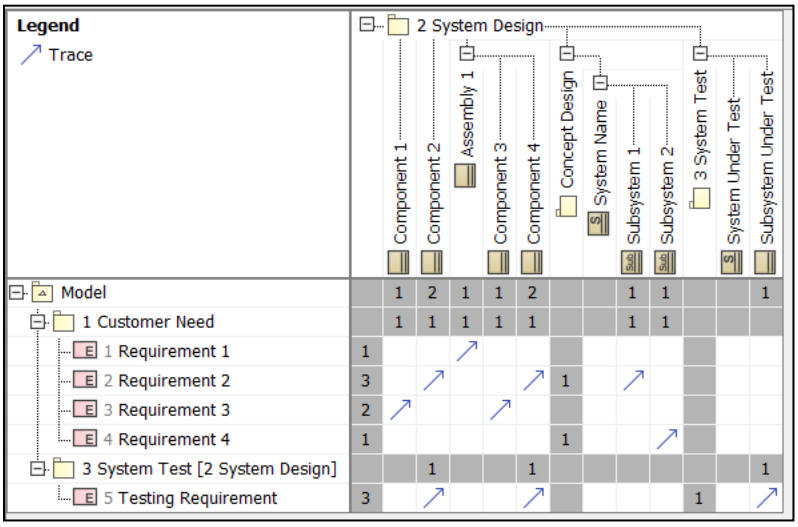

Figure 7: Requirements Traced to System Components

Each system component or block has essential characteristics associated with it, differentiating it from an identical component or describing its parameters. Requirements are often produced around what is needed by the system and may be directly related to these characteristics. In a modeling tool, this can be added directly into a component block, allowing the table to be automatically populated. The table in Figure 8 shows which parts have inputs to requirements and the characteristics that are exhibited in the system.

#	Name	Part	Traced From
1	Subsystem Under Test	V Test Characteristic = #	E 5 Testing Requirement
2	Component 1	P Characteristic 1 = #	E 3 Requirement 3
3	Component 2	V Characteristic 2 = # V Characteristic 3 = #	E 2 Requirement 2 E 5 Testing Requirement
4	Assembly 1	P component 3 : Component 3 P component 4 : Component 4	E 1 Requirement 1
5	Component 3	V Characteristic 4 = #	E 3 Requirement 3
6	Component 4	P Integration Characteristic = #	E 2 Requirement 2 E 5 Testing Requirement

Figure 8: System Characteristics Related to Requirements

By identifying the important characteristics, verification activity data to be collected can be more easily determined. The actual values of the characteristics are dependent upon the solution (some of which may be known up front when using commercial off the shelf products) and the outcome of system testing at its various levels.

Focusing on the verification activities is the next logical step after having created relationships from the family of objects used in the systems engineering factory (requirements, verification method, and system architecture). The component tracing, characteristics, and verification methods are used in the class to determine the verification activities required for the system and at different levels of system testing (component, subsystem, etc.).

Depending how a program determines which verification activities are necessary may determine the order in which to conduct the final coupling exercises. Figure 9 shows the requirement that will be verified at a specific verification event. The components required or system parts being verified at the event are shown in Figure 10. These elements are traced to the verification events that will take place. The completed matrices allow for ensuring all requirements have corresponding verification events and what components are required at the event.

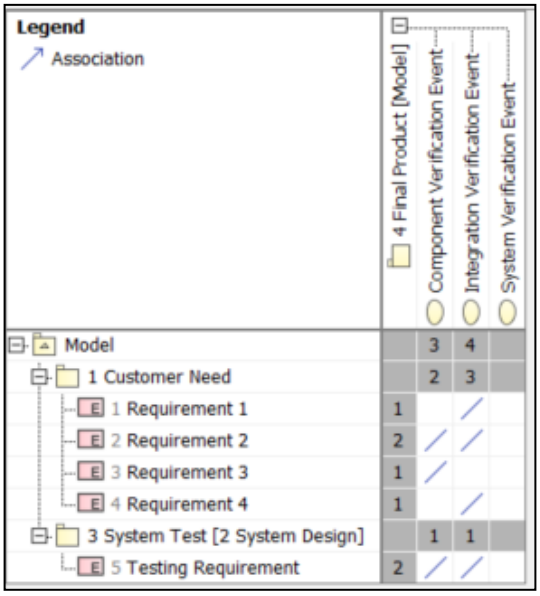

Figure 9: Requirements Traced to Verification Events

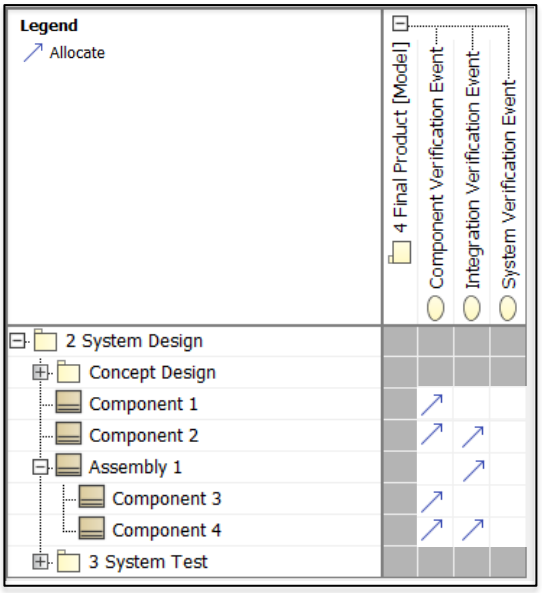

Figure 10: Components Traced to Verification Events

4.2 Verification Information Table

The views and model coupling described in the previous section leverages the inputs and outputs derived in the system verification realm from Section 3.1. This set of views can provide value on their own for detailed analysis of the system, but the aggregation of the coupling shown between the elements produces the verification information table at the heart of this thesis shown below.

Requirement Verification Status: Pass Fail Untested

#	Name	Verify Method	Owned Attribute	Documentation	Traced To	Active Hyperlink	Verified By	Verification Date	Time Phase
1	Requirement 1	Demonstration	: Integration Verification Event	Integration of assembly successfully demonstrated.	Assembly 1 Integration Characteristic = # Characteristic 4 = #	Demonstration Proced http://DemonstrationProc...	Demonstration Report	MM/YYYY	Final Implementation
2	Requirement 2	Analysis	: Component Verification Event : Integration Verification Event	Component analysis within requirement constraints as applicable in required standard. Integration level analysis within tolerance levels shown in Analysis Report 2.	Component 2 Subsystem 1 Component 4 Characteristic 2 = # Integration Characteristic = # Characteristic 3 = #	http://IEEE Standard ###	Analysis Report 1 Analysis Report 2	MM/YYYY	Final Implementation
3	Requirement 3	Test	: Component Verification Event	Component level testing unsuccessful for Component 1, documented in Test Report 1.	Component 3 Component 1 Characteristic 1 = # Characteristic 4 = #	Test Report 1 Component Threshold		MM/YYYY	Design Options [Clas
4	Requirement 4	Inspection	: Integration Verification Event		Subsystem 2	mdel://_19_0_4_58901ef.. Inspection Form		MM/YYYY	Conceptual Design [Clas
5	Testing Requirement	Test	: Component Verification Event : Integration Verification Event	Full testing report for components shows adherence to testing requirements. Subsystem test unit needs to be tested.	Subsystem Under Test Component 2 Component 4 Characteristic 2 = # Integration Characteristic = # +Test Characteristic = # Characteristic 3 = #		Testing Report	MM/YYYY	Test Hardware [Clas

Figure 11: Verification Information Table

This table enables the visualization of the summary of information from the coupling created in the model for the verification data produced and described within this paper. The colors for each row correspond to the requirement's verification status, i.e., whether it has passed, failed, or is currently untested. This provides quick feedback for how the system's performance is compared to the verification criteria given. The first column identifies the requirement, the standard the system needs to adhere to and the description of the need or rules of the system. Each one of these rules has an associated verification method (demonstration, inspection, analysis or test) as made common by the systems engineering field. The next column identifies the name of the verification activity that the requirement was evaluated at with a corresponding *Summary* column. The verification activities are represented by events named after common levels of system verification [30]. This can be easily extended to other types of verification activities and environments specific to the system being produced. The activities used here are Element Verification, Integrated System Verification, and Final Verification. The system components (decomposition or architecture) and supporting equipment or resources being used at the event or evaluated for validation of the system populate the "Traced To" column. The *Supporting Documentation* column is shown populated with hyperlinks, test procedure documents, or links internal to the model for test outcome values or other modeling views relating to or supporting planning and execution of the verification activity. The *Verified By* column shows verification evidence of the requirement, usually presented in a report produced by the program. This is the final evidence showing system compliance and validation to the requirements and is followed by the date the requirement was verified or last tested. Lastly, the *Time Phase* is included to indicate during what point in the system lifecycle the requirement was evaluated at. This helps to provide continuous expected or actual system performance throughout product development. These columns were chosen to represent the variables utilized within the thesis and summarizing the results.

5. Results and Applications

To more concretely demonstrate the application of this verification framework for a project or product, I will use an example to show information gain from the coupling discussed in the previous section. This example uses the power subsystem of a satellite as a white box, shown by the system's conceptual Block Definition Diagram (BDD) in Figure 12 below.

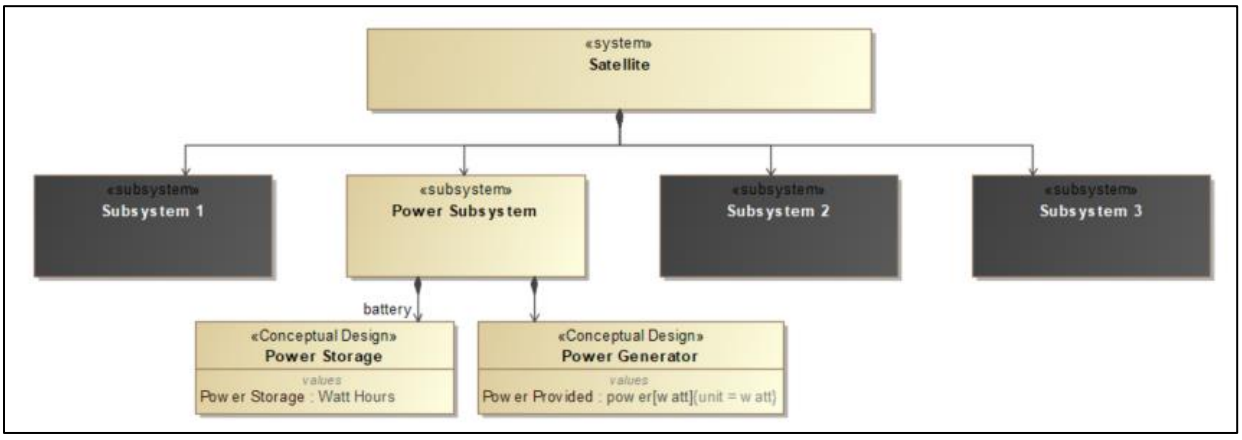

Figure 12: Satellite System Conceptual BDD

The requirements used for the example are arbitrary with reasonable statements that may be encountered for a realistic design and are as follows [7]:

> P1 *Lifecycle System Power*- The satellite shall maintain power throughout its lifecycle.
>
> P1.1 *System Power*- The satellite shall retain 10 years of power or be able to restore its power source.
>
> P2 *Power Supplied*- The satellite shall supply power for all required operations.
>
> P3 *Power Reserve*- If the power source is rechargeable, it shall achieve a storage of no less than 120 watt hours before a daily down cycle starts.

The information gained over time is presented by using generic development phases to represent time points/progression on a project. These markers will be the Conceptual Design, Design Option, Test Hardware, and Final Implementation. As the project is developed, the following views reflect the current state of meeting the requirements.

To round out the example content I have used the same generic verification activities: System Verification Event, Integration Verification Event, and Conceptual Verification Event. I have added Battery Lifecycle Testing, Solar Panel Quality Testing, and System Battery Analysis to illustrate connections to realistic examples of testing for the requirements.

Finally, I will show the model's response to change using three scenarios that follow in the next subsections.

5.1 Product Example

The conceptual design is presented again in Figure 13 with the conceptual system as rows. Below the subsystem level, the parts are allocated to the concept they are fulfilling.

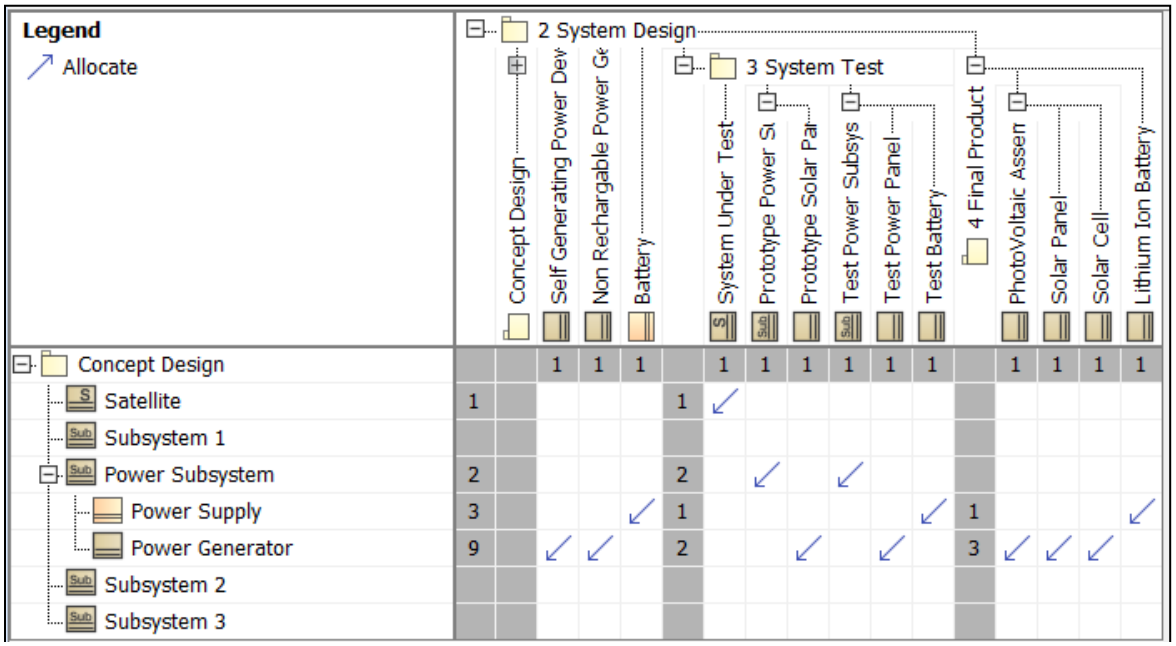

Figure 13: Satellite System Concept Allocated to Design Implementation

In the modeling tool, this indicates that it is a more specific implementation of the design. To show progression of blocks throughout the system lifecycle, the design options are created as blocks in the columns with elements that will fulfill the system need for power generation and power supply such as *Self Generating Power Device* or *Non-Rechargeable Power Generation*. Several folders are seen to show the growth of the matrix or product's maturity by the blocks nested in system test resources and the final product.

We see via the matrix that the need for a power supply will be fulfilled by a battery included in the system design. As the product develops, the system will be tested with test components and a prototype will be built. These same blocks, along with the Conceptual Design from the customer, are shown again in Figure 14 with traces to the requirements. The progression of blocks available shows where the prototype and test elements can be assessed against the requirements for initial compliance. The final product components to be assessed against the requirements are shown by the arrows indicated in the final four columns.

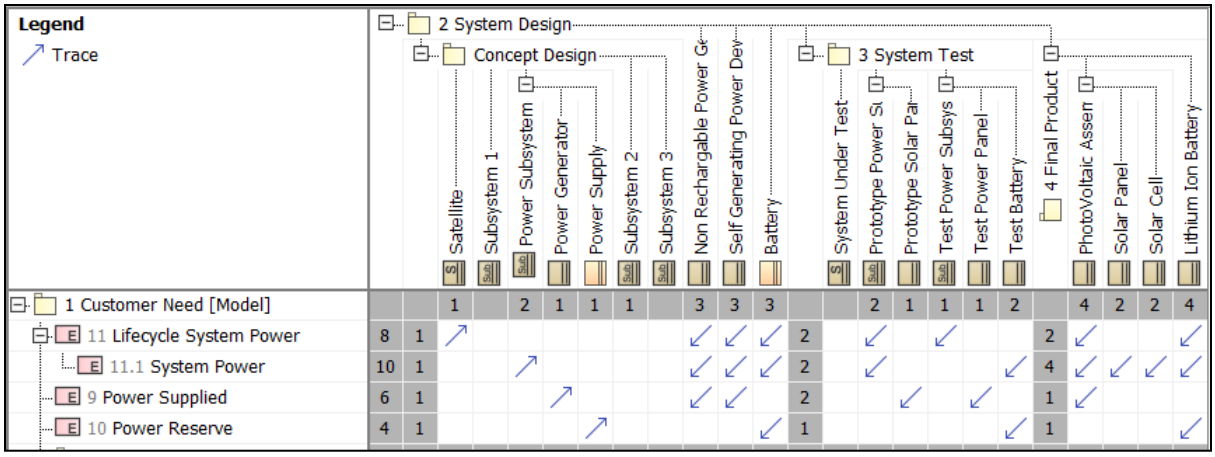

Figure 14: Satellite Requirements Traced to System Components

These traces and the properties associated with the blocks can be transformed into a table as shown in Figure 15 below. In our invented system here, we chose an arbitrary power load (from communication devices among other possibilities) of 10 watts. Therefore, given an approximately 12 hours on and 12 hours off cycle (based on available sunlight to a satellite revolving the Earth), the parts chosen to fulfill the requirement of accumulating and storing 120 watt-hours of energy per light cycle is a battery that can hold at least 120 watt-hours and a solar array that can provide at least 20 watts. Together, these parts will allow, at minimum, the accumulation of the required 120 watt-hours of energy for the dark cycle from a single light cycle while starting from a fully depleted battery.

Name	Part	Traced
PhotoVoltaic Assembly	Total Power = 20 Watts	Lifecycle System Power System Power Power Supplied
Solar Cell	Power Source = 2 Watts	System Power
Solar Panel	Panel Power = 10 Watts	System Power
Lithium Ion Battery	Power Storage = 120 Watt Hours	Lifecycle System Power System Power Power Reserve

Figure 15: Satellite Component Table with associated Characteristics and Requirements

Turning to the verification activities, the presentation of the requirements and system elements in matrix form can be used to see when and where requirements will be verified and what components will be used to verify the requirements. A representation of this is provided using the Satellite example's verification activities, sample requirements, and the final product satellite components show in Figure 16 and Figure 17. These matrices show us the Solar Panel Quality test will verify the Power Reserve requirement. This verification event requires the solar panel from the Photovoltaic Assembly for the testing. Cumulatively, this information can be used to plan for the verification activities and direct the collection of data.

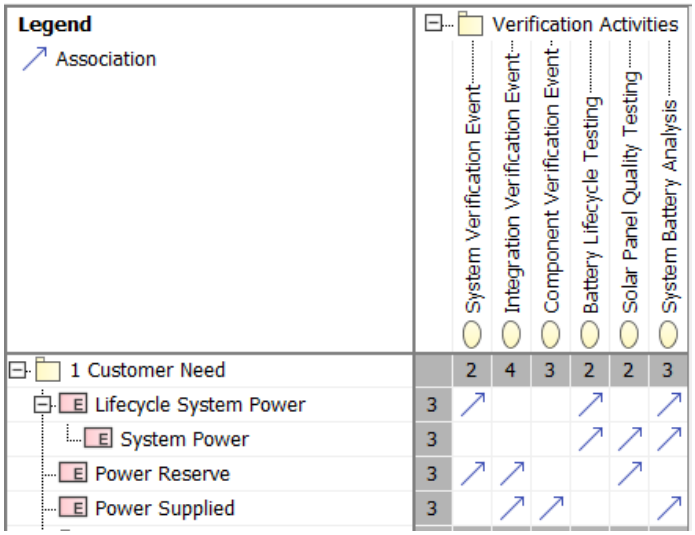

Figure 17: Example of Satellite Requirements Associated with Verification Events

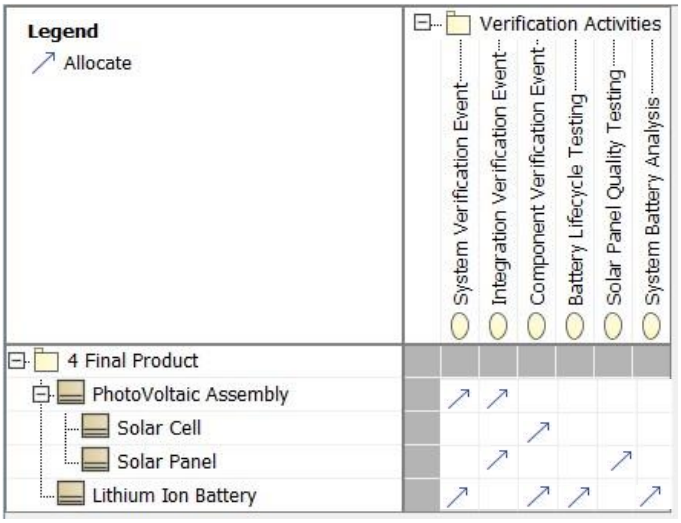

Figure 16: Example of Satellite Components Allocated to Verification Events

The end product created from the smaller views is the verification information table capturing the requirement verification status (Figure 18), method of verification and corresponding event(s), the resources used, and the verification output product. Upon first glance we immediately see that two of the requirements are fully verified and two requirements have not yet been fully verified. All the requirements have been partially verified due to the completeness of reports populating the *Verified By* column. The reports listed in this column are artifact blocks which will open the report when it is clicked on in Cameo.

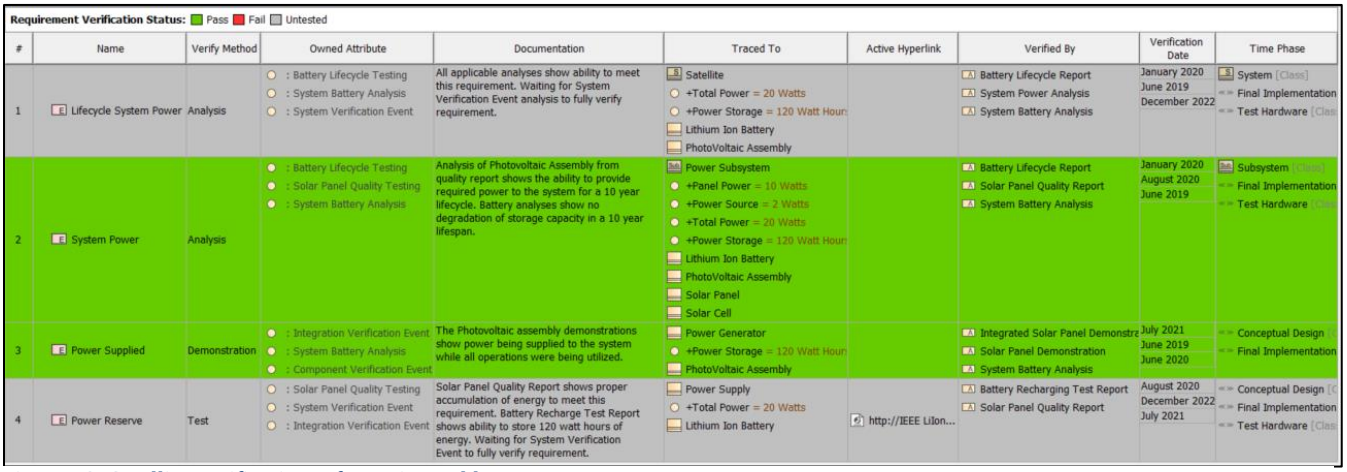

Figure 18: Satellite Verification Information Table

For this arbitrary example, I will use the second row of the verification information table to describe verification of the System Power requirement. Clicking on the requirement block in the table, the specification of the requirement shows up as shown in Figure 19. This window shows us detailed information about the requirement.

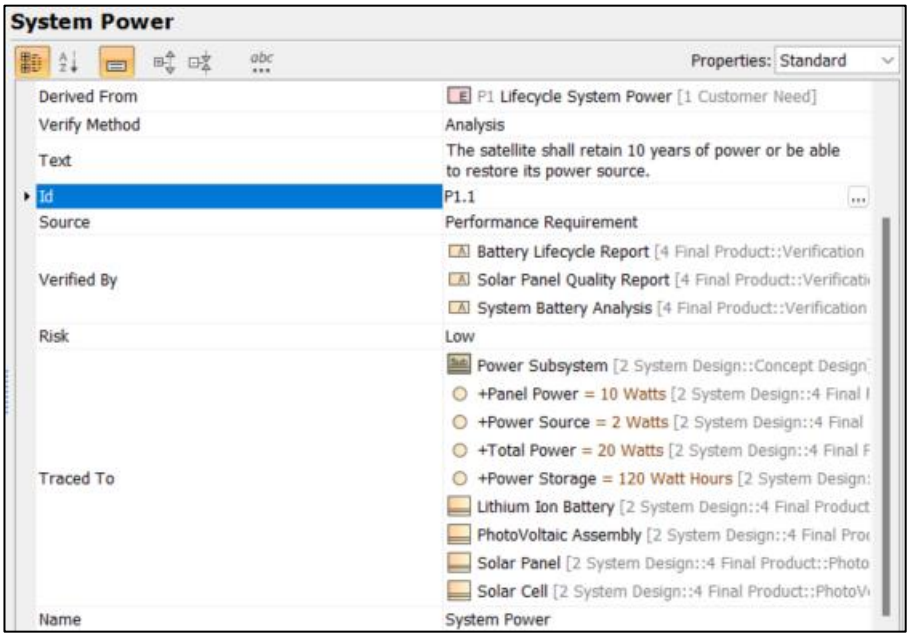

Figure 19: Requirement Specification Window of Example Satellite Requirement

We can see that this requirement is derived from P1 Lifecyle System Power Requirement, is verified by Analysis, and the requirement is that, "The satellite shall retain 10 years of power or be able to restore its power source". There is low risk associated with the requirement and the components that contribute to meeting the requirement, along with their characteristics, are shown in the *Traced To* field. The requirement verification artifacts are the Battery Lifecycle Report, Solar Panel Quality Report, and System Battery Analysis.

Returning to the verification information table, the *Time Phase* column shows that the components of the power subsystem were part of the "Final Implementation" phase intended to be representative of the final product. Earlier in the system development "Test Hardware" would have been the only item displayed to show preliminary compliance towards meeting the requirement during the initial testing point of product development.

Three verification events were completed to show verification of this requirement as listed in *Owned Attribute*. The *Documentation* field provides a summary of the requirement's verification stating, "Analysis of Photovoltaic Assembly from quality report shows the ability to provide required power to the system for a 10 year lifecycle. Battery analyses show no degradation of storage capacity in a 10 year lifespan". The System Battery Analysis event listed has a corresponding report that was completed in June of 2019. This battery analysis report would have evidence towards the battery being a rechargeable source of power for the system. The Solar Panel Quality Report and Battery Lifecycle Report, completed after their corresponding events in 2020 would include analyses demonstrating the reliability of the

hardware over a 10 year lifespan. The solar panels are the source of continually collecting power for the system. With all three pieces of evidence, the requirement is shown to be fully verified.

5.2 Use Case: Requirement Change

The first use I am presenting is the framework utility when a requirement is changed. The example requirement P3, *Power Reserve*- "If the power source is rechargeable, it shall achieve a storage of no less than 120 watt hours before a daily down cycle starts", is changed to "The power source shall achieve a storage of no less than 180 watt hours before a daily down cycle starts". Cameo can trace changes multiple ways in the model. One way is using project compare which provides an output, highlighted in Blue in Figure 20, to represent modifications to elements in the model.

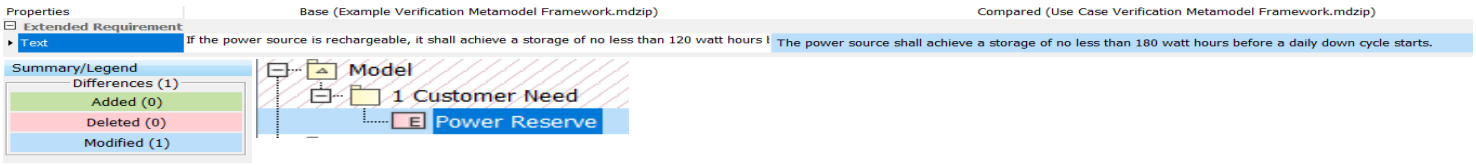

Figure 20: Project Comparison after a Requirement Change

The changed requirement can be pulled onto a requirement diagram and all related elements displayed, Figure 21. Investigating the element further, we see what the inputs were to the requirement's verification. This requirement's documentation shows that in addition to looking at the battery's capacity we would need to look at the solar panels' capability to accumulate the new charge requirement. The Integration, System, System Battery Analysis, and Solar Panel Quality Testing verification events may need to be repeated.

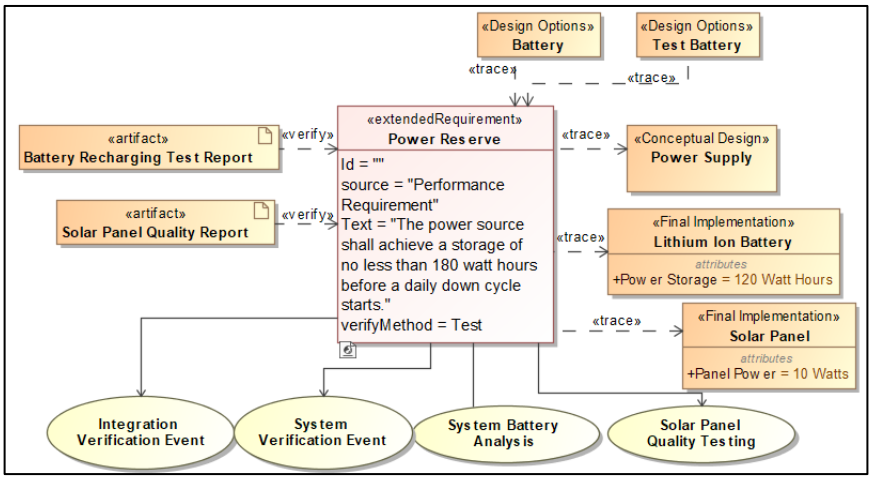

Figure 21: Requirement Verification Elements Displayed for Analysis after Requirement Change

Figure 21 also represents the essential model relationships in the verification framework to produce the information table; the backbone of how the model elements must be coupled.

5.3 Use Case: Second Verification Event

The second application shown is the case where a verification event would be replanned or repeated. In a document-centric environment, the process would include locating and retrieving the original test plan and procedures and updating it to reflect the new test. A separate document would contain test assets and a third the schedule. The requirements team would provide the updated requirements to be changed in all the documentation. After, the test the report is written and archived.

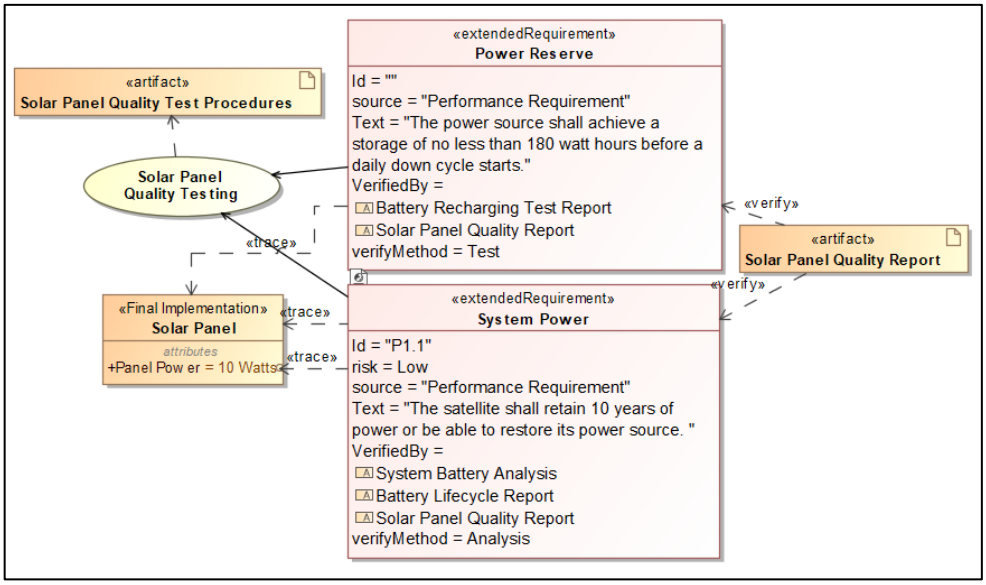

Figure 22: Verification Event Resources Displayed

Locating the verification event of interest in the model, the related elements can again be displayed showing the requirements and resources associated with the original verification event element (Figure 22). All the documentation associated with the event can be opened from this page. This allows complete and immediate access to everything associated for planning and full contextual understanding of the verification event's place among system verification and to aid in replanning of the activity.

The document-based approach would not give this visibility or provide quick access to the associated documents. One would need to look at the report itself to see what documents were referenced and locate those to get the associated information. The network of documentation associated with the event and the additional verification events that may rely on the results of this one for the requirement's complete validation may be overlooked or not have the level of visibility easily seen here.

5.4 Use Case: Component Reuse

The third application is transferring a part from a system model to a new development effort. The Lithium-Ion battery has many uses across different types of systems. The model aids in reuse of objects across models and supports a modular approach to system design. Using the associated testing evidence and requirements the battery has been evaluated against or the standards required of it (Figure 23), we can bring these objects into a new model to possibly reduce future testing.

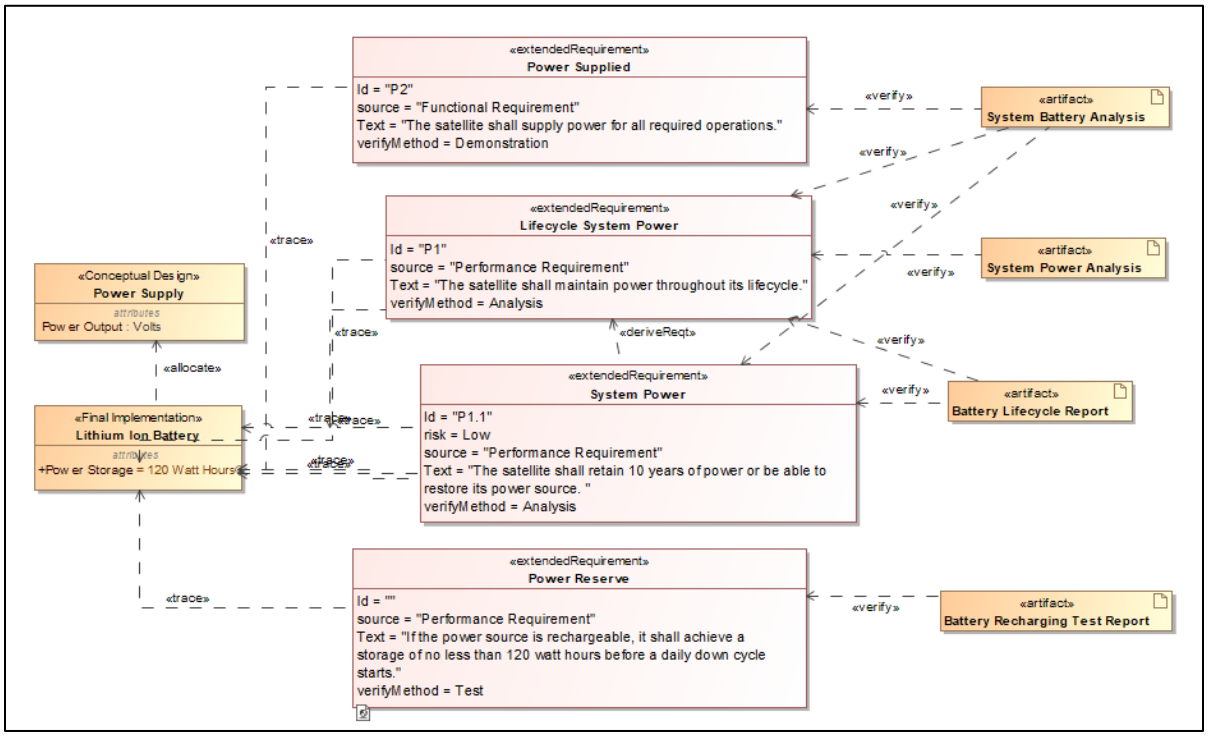

Figure 23: Battery Artifacts in Cameo for Component Reuse

The elements in Cameo can be copied into a new model and the verification status updated based on the new system's requirements. The artifacts still associated with the battery's documentation are transferred into the new model and await verification against the new system's requirements. The allocation to *Power Supply* shows what conceptual purpose the battery served so it can be easily allocated to the new system's architecture, as shown in the table in Figure 24.

Name	Component Documentation	Requirement and Verification Event
Lithium Ion Battery	Battery Recharging Test Report Solar Panel Quality Report System Battery Analysis Battery Lifecycle Report	Association[Power Reserve -> System Verification Event] Association[Power Reserve -> Solar Panel Quality Testing] Association[Power Reserve -> Integration Verification Event] Association[System Power -> Battery Lifecycle Testing] Association[System Power -> Solar Panel Quality Testing] Association[System Power -> System Battery Analysis]

Figure 24: Allocations and Documentation associated with Battery Reuse Component

6. Discussion

To arrive at the verification information table presented as the output to the proposed verification framework, the system lifecycle was considered to encompass all the variables related to system verification. This verification information display produced by the model expands the VCRM allowing for layperson use of the tool, and for the identification of issues or gaps earlier in the system lifecycle. The object-oriented approach to implementing this framework in a model-based environment ensures proper design considerations are taken to allow the framework to be robust and adapt as industrial digital transformation evolves. This leaves room for the specific program completing the task to customize how the objects are handled, verified and implemented into the non-changing areas of verification from the pattern [29].

The system verification information view (shown in Figure 11) is an expansion of the VCRM which is usually comprised of the system requirements and their verification methods only. The verification information table framework presented here uses the requirements and verification method as input objects, along with the system design representation, to build a more complete view (model relationships shown in Figure 21). In the model, the verification method is a single property of the requirement object. Showing the meaningful relationships for the context of preparing for, completing, and understanding the outcome of a verification event is absent in the standard VCRM product. Therefore, coupling the requirements with their associated system elements is the first logical step in understanding how the requirement will be verified. Modeling allows for the system components to be tied to requirement objects. These system component objects contain their own properties or characteristics describing the component and system. This coupling provides an understanding of what the system is exactly looking to verify. Decomposing the requirements and components to their essential characteristics provides another layer of detail in preparation for system verification. Including this additional layer of information allows the data collected to be known and design trade studies to be completed.

The component blocks in the model can include supplementary coupling in the model to the important elements of the system related to the customer needs, thereby supporting a Modular Open Systems Architecture (MOSA) approach. I used this idea in the first matrix presented to show concept relationships to system components (Figure 6) so multiple parts could be shown representing the design change over the system development lifecycle. This allows one to envision how the matrices will grow over time and change as the system is realized. This exercise provides a good reference point to anchor elements to the customer need/concept and to visualize the resources in the system which may evolve during the design process. The goal of MOSA is to use generic concepts or system elements to represent the system in a modular fashion for ease of replacement or relations to existing concepts. The maintenance of this matrix would be necessary if looking to change components in a system or across systems.

The last category of information included is at which verification event(s) the requirement will be verified. Creating the verification events and requirement relationships, or, if one prefers, the events and corresponding system elements to be evaluated, provides the complete traceability path within the model for all the verification variables. In the model, either

path creates the coupling needed to provide elements crucial to verification planning in the view. The resources (requirements to be verified, data to be collected, and system elements being evaluated) provide the visibility and context to understand the system's verification for the customer, test director, and engineers involved.

Verification activities and planning should also include related data/documents and the standards associated with the event for its preparedness. The verification activities are the methods for how the requirement will be tested given the object(s) being verified and the additional resources and test facilities the test requires. The performance data is how the system is behaving during one of these activities or tests. These outcomes are the ultimate documentation to show that the system is performing as intended and in the way the customer desired.

In conclusion, the verification information view provides all information related to system verification disassociated from the individual contractor's verification strategy. When implemented early in a project, it provides continuous verification compliance status and minimizes the management required of systems engineers by using a collaborative environment to provide the information. The model-based environment features modularity to customize the view for the system designers and other stakeholders. The structure is reusable and allows new requirements and components to be added-in by identifying the areas of coupling that must follow for completeness and inclusion. The framework captures system component maturity/information and provides detailed system information over time due to the coupling of the model inputs.

6.1 Benefits of the Model-Based Verification Information Framework

The verification framework presented, when instantiated in a model-based tool, can provide numerous views of aggregate information for one or all verification events. These would be usable by the customer, the systems engineers and development engineers, and the system testing group during all points in the system development lifecycle.

Including the interfaces suggested in the AFP for information flow would allow the engineers on the program, or at a certain company, utilize their unique verification strategy within the process. This freedom is essential to the design process and lets the engineers adapt to changes in industry. Changes in workflow and processes are seen in engineering methods, industry standards, and between companies or parties doing the work.

The information provided assists in viewing project health in terms of current design maturity, ability to meet requirements, sharing of system knowledge cross-functionally, planning for testing events, and data management. Using this framework in the model environment allows for a collaborative workspace for all system information. This aids in the information handoff, document localization, and format standardization. Thus, decreasing format variability and time spent on information gathering and documenting.

The matrices created to complete coupling within the model provide insight to some of the smaller views that can be utilized by the system stakeholders. I will refer to these views as Product Maturity (Figure 6 and Figure 8), Requirement Traceability (Figure 7), and Verification

Event (Figure 9 and Figure 10). The verification information table itself (Figure 11) can be manipulated easily in the model-based environment to limit or expand what information is showing for the intended recipient.

Product Maturity is seen by the trace from system concepts to system design. Using the system's logical blocks (components deduced as being part of the system based on the customer's need/description) traced to the current actualized components allows the customer to see adherence to requirements during the developmental phases and the engineers are able to trace the system's development back to the requirements as the hardware is chosen. This provides a type of taxonomy for the program to use to communicate with the customer and show design fulfillment in context understood by the customer's point of view. In the Satellite example, during the design phase there may be a choice between using two different assemblies to fulfill power generation. They may each be evaluated against the system requirements but eventually the implemented design is what needs to be verified. Using a commercial off the shelf part or pre-designed component increases the maturity of that part of the system. Known components may be introduced into the design early, but the verification framework places it closer towards being a verifiable component by adding its characteristics. Seeing the components maturity levels overall can help identify how close the system is towards a technical readiness level or towards the readiness the unique gates a development group may use.

Requirement Traceability is seen by the links between requirements, components and their characteristics, and eventually the verification events. Adding the component characteristics may help increase the level of detail produced when analyzing where the requirement ties into the system, as shown in Figure 25. I have added this level of detail to enhance Figure 7 based off the table in Figure 8. If a requirement is directly related to a component's characteristic, then during trade studies the estimated values can be displayed to see if the suggested design will meet the requirements.

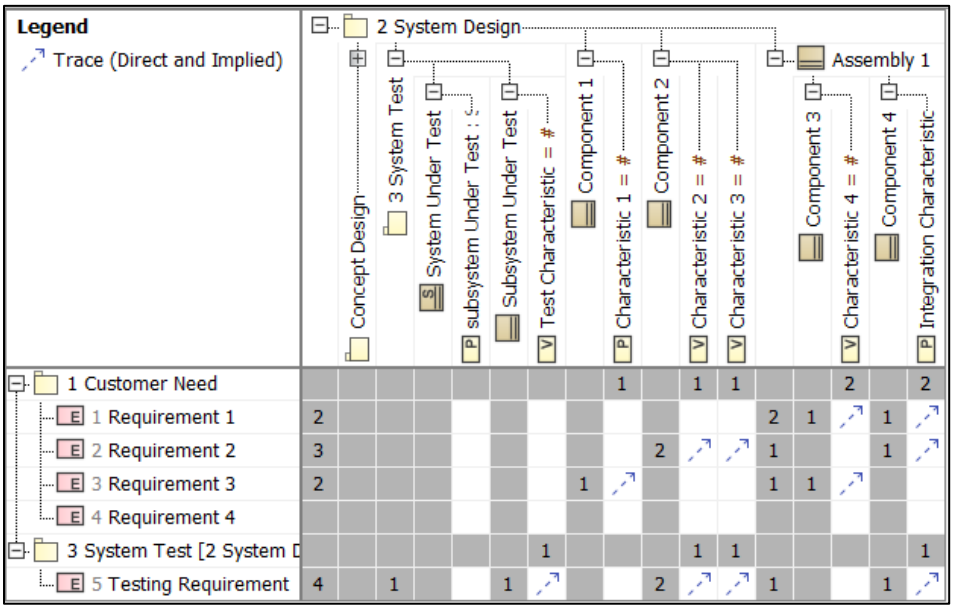

Figure 25: Requirements Traced to Component Characteristics

The benefit to the systems engineers and design engineers by tracing to Verification Events provides products showing incremental information for how the system will be verified as it is determined. If the model-centric environment is already in place, a hand off between systems engineer and the verification process is not required as it can be provided directly to the customer to allow for full visibility. From the resources traced to the verification events, the views can be utilized for test plan creation and event preparation and helps compile and trace artifacts as event outcomes are completed. This view can quickly show the frequency of testing planned and how many opportunities are available to verify a particular requirement. The level the requirement could be tested at can be captured to reduce system testing or create a justification for additional testing needs. This capability can catch if certain components or requirements do not have a plan for testing, where the system is tested, and what resources are required for the testing.

The final verification information table helps systems engineers to ensure compilation of all system information in one spot as the design progresses and outcomes are completed. It demonstrates system verification and validation, capturing the levels the system is verified at and can include links to the testing evidence. One is able to determine what tests or functions will be looked at during a verification activity using the related parts of the system for a specific verification activity at any point in the program. This is a more detailed approach that can be easily manipulated to produce multiple views of VCRM, see testing history and the product configuration used.

6.2 Constraints

The implementation proposed is a very basic implementation of verification integration throughout the system lifecycle. Further development and industry buy-in across all fields of engineering is required to make this a reliable and complete network of information. Information is gathered from many sources, each which is not always provided from external sources in the same format. The way around this currently is to create hyperlinks within the model fields to the resources. This, however, makes the tool more of a visual search database instead of a complete source for all related information.

The verification information table does not provide an easy way of determining engineering decisions that need to be made and a human-in-the-loop is still required to interpret the results and do a complete search for gaps or information needed. Additionally, since these applications are not yet standard in industry, it becomes another tool that needs to be integrated into programs and have users trained in its population [31]. Maintenance of the tool information is also a manual process without integration into the testing environment or processes that include ensuring the products are added in the model as verification events are completed. The current state is the idea to make systems engineering products more valuable and show its potential value with continued development using these tools. Lastly, as this is just a method for visualization, it in itself does not provide any novel information to any of the parties involved.

6.3 Further Applications

Provided the structure discussed is applied on a project, several avenues open to continue investigation and application of the verification process in model-based tools for completeness. The outcome of this strategy provides opportunity to implement verification activity planning

paths and continuous confidence towards meeting systems requirements. More information can be included depending how much development the designer or program chooses to include in the model. As MBSE becomes more prevalent, this will naturally become available.

Compatibility with Dynamic Object Oriented Requirements System (DOORS) is available to leverage the work done in this modeling tool along with integration to other engineering models. This allows for more detailed views to be developed and reduce handoffs between data products being created.

More data can be added. For example, one might include testing resources and the program schedule for the activities linked to the components/activities to develop views for wider use or associate risk status with the requirement from preliminary analysis. Additionally, further levels of requirement status can be built into the framework by including an Untested category, a Compliance category, and a Verified category and associating statuses for each. This is an initial approach to providing confidence in meeting requirements as early as design decisions are made, testing occurs, and time passes. This can be captured in the model-based environment by utilizing its metrics capabilities and communicating progress to the customer. Categorizing product families within the model can capture configurations for the desired use. It could allow for versioning of the system in response to new upgrades or for capturing prototype evolution during development. Seeing the component's maturity levels can help identify the technical readiness level of a system or the readiness of the unique gates a development group may use.

As we head more towards model-based environments, levels of automation and computer aided validation techniques can be integrated to reduce human-dependent checks. The theoretical implementation has already been researched for automation of the verification and merely needs to be integrated with a MBSE tool [30,31,34].

7. References

1. Sousa-Poza AA, Keating CB, Kovacic S. Systems Engineering: Evolution and Challenges. Int J Syst Syst Eng [Internet]. 2014 [cited 2022 Aug 16];5(4):379–99. Available from: https://www.researchgate.net/publication/273457341

2. McDermott T, Henderson K, Aken E Van, Salado A, Bradley J. Measuring the Role of Digital Engineering: It's a Journey, Not a Number. AIRC Perspect [Internet]. 2022; Available from: https://digitalcommons.odu.edu/emse_fac_pubs/139

3. McDermott T, Salado A, Aken E Van, Henderson K. A Framework to Categorize the Benefits and Value of Digital Engineering. 2021; Available from: https://dair.nps.edu/handle/123456789/4375

4. Baker L, Clemente P, Cohen B, Permenter L, Purves B, Salmon P. Foundational Concepts for Model Driven System Design.

5. Salado A. Efficient and Effective Systems Integration and Verification Planning Using a Model-Centric Environment. INCOSE Int Symp [Internet]. 2013;23(1):1159–73. Available from: https://onlinelibrary.wiley.com/doi/full/10.1002/j.2334-5837.2013.tb03078.x

6. Süß JG, Leicher A, Weber H, Kutsche RD. Model-Centric Engineering with the Evolution and Validation Environment. Lect Notes Comput Sci (including Subser Lect Notes Artif Intell Lect Notes Bioinformatics) [Internet]. 2003 [cited 2022 Aug 16];2863:31–43. Available from: https://link.springer.com/chapter/10.1007/978-3-540-45221-8_4

7. Saeed N, Elzanaty A, Almorad H, Dahrouj H, Al-Naffouri Y, Alouini M-S. CubeSat Communications: Recent Advances and Future Challenges.

8. MBSEworks.com. MBSE FAQ: How does MBSE enable Requirements V&V? [Internet]. [cited 2022 Aug 20]. Available from: https://mbseworks.com/mbse-faq/how-does-mbse-enable-requirements-verification-and-validation.html

9. Gaignebet A, Chapurlat V, Zacharewicz G, Richet V, Plana R. A Model Based System Commissioning Approach for Nuclear Facilities. Sustain 2021, Vol 13, Page 10520 [Internet]. 2021 Sep 22 [cited 2022 Aug 17];13(19):10520. Available from: https://www.mdpi.com/2071-1050/13/19/10520/htm

10. Eppinger SD. Model-Based Approaches to Managing Concurrent Engineering. http://dx.doi.org/101080/09544829108901686 [Internet]. 2007 Jan 1 [cited 2022 Aug 17];2(4):283–90. Available from: https://www.tandfonline.com/doi/abs/10.1080/09544829108901686

11. Nassereddine H, Jazzar M El, Piskernik M. Transforming the AEC Industry: A Model-Centric Approach. 2020 [cited 2022 Aug 16];76. Available from: https://doi.org/10.3311/CCC2020-076

12. Akundi A, Mondragon O. Model based systems engineering—A text mining based structured comprehensive overview. Syst Eng [Internet]. 2022 Jan 1 [cited 2022 Aug

17];25(1):51–67. Available from: https://onlinelibrary.wiley.com/doi/full/10.1002/sys.21601

13. Henderson K, Salado A. Value and Benefits of Model-Based Systems Engineering (MBSE): Evidence from the Literature. Syst Eng. 2021;24(1):51–66.

14. Marcos V. Linhares, Romulo S. de Oliveira, Jean-Marie Farines FV. Introducing the Modeling and Verification process in SysML [Internet]. 12th IEEE Int. Conf. on Emergin Technologies and Factory Automation. 2007 [cited 2022 Aug 20]. Available from: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.936.764&rep=rep1&type=pdf

15. Wheeler R. Rich Wheeler's Thoughts on SE, BA, RM, BPM, PM, and Other Nerdly Stuff: The Verification Cross-reference Matrix (VCRM) [Internet]. 2011 [cited 2022 Aug 20]. Available from: http://richwheeler.blogspot.com/2011/06/verification-cross-reference-matrix.html

16. Laing C, David P, Blanco E, Dorel X. Questioning Integration of Verification in Model-Based Systems Engineering: An Industrial Perspective. Comput Ind. 2020;114.

17. Honour E. Verification and Validation Issues in Systems of Systems. Electron Proc Theor Comput Sci [Internet]. 2013;133:2–7. Available from: http://arxiv.org/abs/1311.3626

18. Estefan JA. Survey of Model-Based Systems Engineering (MBSE) Methodologies. 2008;

19. About the OMG System Modeling Language Specification Version 1.5 [Internet]. [cited 2022 Nov 26]. Available from: https://www.omg.org/spec/SysML/1.5

20. Montgomery PR. Model-Based System Integration (MBSI) – Key Attributes of MBSE from the System Integrator's Perspective. Procedia Comput Sci. 2013 Jan 1;16:313–22.

21. European Cooperation for Space Standardization (ECSS). Verification. ECSS-E-ST-10-02C. 2009;(Second).

22. European Cooperation for Space Standardization (ECSS). Testing. ECSS-E-ST-10-03C. 2012;(Second).

23. NASA. Systems Engineering Handbook. Rev1 ed. Vol. NASA/SP-20. 2007.

24. Selvy BM, Claver C, Angeli G. Using SysML for verification and validation planning on the Large Synoptic Survey Telescope (LSST). https://doi.org/101117/122056773 [Internet]. 2014;9150:236–48. Available from: https://www.spiedigitallibrary.org/conference-proceedings-of-spie/9150/91500N/Using-SysML-for-verification-and-validation-planning-on-the-Large/10.1117/12.2056773.full

25. MacCalman A. U.S. Army Test and Evaluation Command (ATEC) Digital Transformation (DX) Strategy [Internet]. AETC; 2020. Available from: https://apps.dtic.mil/sti/citations/AD1168229

26. Gomaa H. Software Modeling and Design: UML, Use Cases, Patterns, and Software ... - Hassan Gomaa - Google Books [Internet]. [cited 2022 Aug 17]. Available from: https://books.google.com/books?hl=en&lr=&id=TqZi-hAX17YC&oi=fnd&pg=PR7&dq=Gomaa,+H.+(2011).+Software+Life+Cycle+Models+a

nd+Processes.+In+Software+Modeling+and+Design:+UML,+Use+Cases,+Patterns,+and+Software+Architectures+(pp.&ots=4WWaG78tNC&sig=o_gyBMMVRiy

27. Scukanec SJ, van Gaasbeek JR. A Day in the Life of a Verification Requirement. INCOSE Int Symp [Internet]. 2010 Jul 1 [cited 2022 Aug 16];20(1):2524–42. Available from: https://onlinelibrary.wiley.com/doi/full/10.1002/j.2334-5837.2010.tb01180.x

28. R. Shishko, R. Aster, R. G. Chamberlain, P. McDuggee LP. NASA Systems Engineering Handbook [Internet]. NASASP-6105. 1995 [cited 2022 Aug 20]. p. 174. Available from: moz-extension://66831555-3a43-45fc-bc58-182c5d93b088/enhanced-reader.html?openApp&pdf=https%3A%2F%2Fadsabs.harvard.edu%2Fpdf%2F1995NASSP6105.....S

29. Shalloway A and JRT. Design Patterns Explained: A New Perspective on Object-Oriented Design. Second. Pearson Education Inc.; 2005. 193–213, 279–294 p.

30. Walsh S, Murphy D, Doyle M, Thompson J, Dunwoody R, Emam M, et al. Assembly, Integration and Verification Activities for a 2U CubeSat, EIRSAT-1. 3 rd Symp Sp Educ Act.

31. Gallini TE. The Resource Management Segment - A Case Study in the Role of Software Architecture in the Systems Engineering Process. INCOSE Int Symp [Internet]. 1999 Jun 1 [cited 2022 Aug 17];9(1):854–60. Available from: https://onlinelibrary.wiley.com/doi/full/10.1002/j.2334-5837.1999.tb00248.x

32. Salado A, Kannan H. A Mathematical Model of Verification Strategies. Syst Eng [Internet]. 2018;21(6):593–608. Available from: https://onlinelibrary.wiley.com/doi/full/10.1002/sys.21463

33. Salado A, Kannan H. Elemental Patterns of Verification Strategies. Syst Eng [Internet]. 2019;22(5):370–88. Available from: https://onlinelibrary.wiley.com/doi/full/10.1002/sys.21481

34. Gharbi A, Fischer OP, Mavris DN. Towards a Robust Computational Solution for the Verification and Validation of Complex Systems in MBSE using Wymore's Tricotyledon Theory of System Design. AIAA Sci Technol Forum Expo AIAA SciTech Forum 2022 [Internet]. 2022 [cited 2022 Aug 16]; Available from: https://arc.aiaa.org/doi/10.2514/6.2022-0094