
# Systems Engineering Guidance Document

**Revision:** 1.0

**Author/Group:** ABEX Lead Systems Engineer

**Publish Date:** 08/21/2022

***Document Classification:*** *Public*

***Distribution:*** *Unlimited*

## Change Log

| Group | Revision No. | Description | Effective Date |
|---|---|---|---|
| LSE | 1.0 | Initialized Document | 08/21/2022 |
| | | | |

## Table of TBDs

| No. | Section | Description | Date |
|---|---|---|---|
| | | | |

# Table of Contents

# Table of Tables

# Table of Figures

# 1 Introduction

## 1.1 Document Description

The Systems Engineering Guidance Document (SEGD) provides essential knowledge and context for the execution of satellite development, verification, and integration activities. Material in this document is referenced by every other knowledge source in Alabama CubeSat Initiative (ACSI) documentation, so it is imperative to understand material in this document as a reference for future use.

The SEGD begins with a discussion of verification by analysis in Section 2. Technical Performance Measures (TPM) and Knowledge Points (KP) are discussed as analysis results, and Domain Knowledge Maps (DKM) are discussed as representations of analysis methodologies. Integration activities are discussed in Sections 3 and 4. A Product Breakdown Structure (PBS) is described as an organization of Integration Points (IP) in Section 3, and integration activities using the PBS are described in Section 4. The purpose of integration is to prepare for Technology Readiness Level (TRL) advancement testing, and TRLs are discussed in Section 5.

## 1.2 Scope of Applicability

Many Systems Engineering (SE) references exist such as the NASA Systems Engineering Handbook, the INCOSE Systems Engineering Handbook, the Systems Engineering Body of Knowledge, and other valuable materials. Each is hundreds of pages long and does not serve as an appropriate primer for new spacecraft engineers. The scope of this document covers SE principles, operations, and techniques specific to the ACSI in a way that does not require students to read hundreds of pages before becoming proficient. ACSI engineers should reference the ACSI Systems Engineering Introduction presentation as their first source of SE knowledge and use this document as a reference when needed.

## 1.3 Reference Documents

The following project and subsystem documents should be referenced as needed.

**Table 1: Relevant project documents.**

| Project Document Name | Description of Document |
|---|---|
| ACSI Systems Engineering Introduction | Included as a pdf in all onboarding documentation. The pdf of the slides for the intro to SE presentation on the Spacecraft Seminar Series. |
| NASA Systems Engineering Handbook | How NASA does SE. Rigorous, but not every part is necessary for small-scale project or those worth less than $1B |
| INCOSE Systems Engineering Handbook | The INCOSE SE manual. Long document with generalized practices. Highly derivative of the Hatley-Pirbhai method (input-process-output) |
| Systems Engineering Body of Knowledge | A good resource on SE best practices for those looking for a practical mix of how NASA, INCOSE, and the DoD approach SE. |

# 2 Knowledge Points and Technical Performance Measures

## 2.1 Overview

Verification by analysis is performed continually on ACSI spacecraft with analysis methodology described in Subsystem Analysis Plans (SAP) and results calculated or modeled in Subsystem Analysis Reports (SAR). The SAP is created once for every satellite subsystem, and the SAR is completed each time there is a review or design change. Every analysis has six parts,

1. Problem Definition
2. Parameters
3. Assumptions
4. Methodology
5. Results
6. Conclusions

When analyzing a system, the ACSI formalizes parameters into Knowledge Points (KP) and Technical Performance Measures (TPM). Methodology is visually represented by Domain Knowledge Maps (DKM).

### 2.1.1 Knowledge Points

Analysis is performed to calculate or model KPs, or parameters needed to understand a system. KPs could be uncontrollable environmental parameters, intermediate calculations on the way to a more important calculation, or TPMs. **All TPMs are KPs, but not all KPs are TPMs**. KPs can be scalar, array, or matrix parameters depending on the KP definition, but not every parameter in an analysis is a KP. Every KP is given a visual representation of its calculation or modeling methodology known as a DKM. The SAP is simply an organization of KPs and their associated DKMs which allow for the execution of the SAP to create the SAR. KPs are represented as ovals on DKMs whereas everything else is a rectangle.

### 2.1.2 Technical Performance Measures

TPMs are methods of tracking the technical performance of a system over time. They help engineers make decisions about a design by capturing the impact of design choices into helpful parameters. TPMs can be defined for components, subsystems, and systems, and system-level TPMs are considered Key Performance Parameters (KPP). TPMs are commonly defined for capabilities, which are functions with a specified level of performance. The mission may specify a certain level of performance through capability requirements, and that level of performance can be calculated or modeled using a TPM. Requirements that cannot be verified by analysis are often verified instead by test, so it is also possible to test for a TPM. A visual representation of KPs and TPMs is shown in Figure 1.

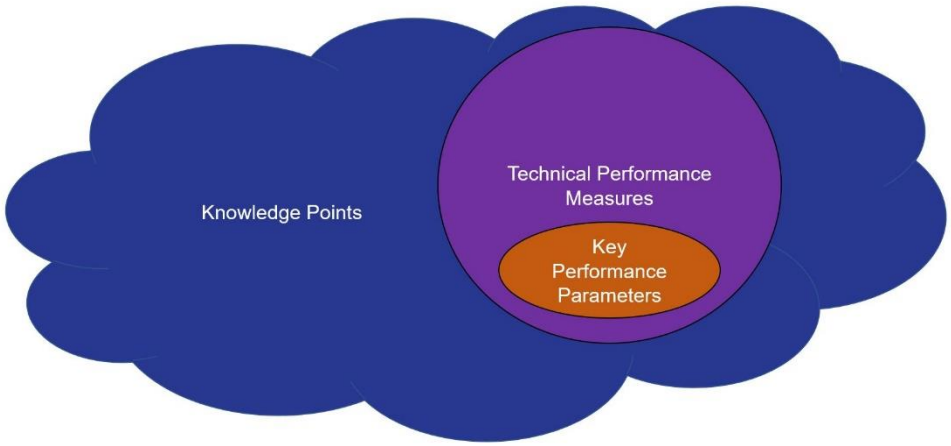


**Figure 1: Knowledge Point Hierarchy**

TPMs have a few defining characteristics that set them apart from regular KPs. They're used to calculate relevant aspects of a design and communicate the value of that design to project stakeholders. TPMs must be tracked and reported each Design Analysis Cycle (DAC). Well-defined TPMs have important features:

- Should be important and relevant to the subsystem design
- Should be relatively easy to measure for reporting
- The performance or knowledge of performance should be expected to improve with time
- A target, threshold, or expectation of uncertainty should be known and if the measure crosses its threshold, corrective action should be known
- The measured parameter should be controllable by the design decisions
- Should be tracked and documented
- Should be tailored for the project

For example, a KP such as an environmental heat flux from the Sun at a given distance may be important and easy to measure, but its performance would not be expected to improve with time. There isn't a target value or threshold for it, and engineers cannot control it by design decisions. Therefore, this KP is something that must be calculated, but it is not a TPM. Formal definitions are provided in Table 2.

**Table 2: KP Types with Examples**

| Parameter Type | Description | Example |
|---|---|---|
| **Knowledge Point** | Any parameter needed to understand a system's environment, behavior, structure, or operations that can be calculated or modeled. Non-TPM KPs are subjectively determined from a total parameter list. | Solar Emission Heat Flux |
| **Technical Performance Measure** | A set of performance measures that are monitored by comparing the current actual achievement of the parameters with that anticipated at the current time and on future dates (NASA Systems Engineering Handbook, 2016) | Subsystem Mass |
| **Key Performance Parameter** | Those capabilities or characteristics (typically engineering-based or related to health and safety or operational performance) considered most essential for successful mission accomplishment. They characterize the major drivers of operational performance, supportability, and interoperability [1] | Spacecraft Mass Margin |

## 2.2 Domain Knowledge Maps

Domain Knowledge Maps (DKM) are visual representations of analysis methodology for a given KP. They describe sources providing calculation or model input parameters, equations or models used to determine the KP, and the methodology order of operations. DKMs are diagrams/figures used for both onboarding and analysis execution and are the central focus of Subsystem Analysis Plans (SAP).

### 2.2.1 Formal Language of DKMs

The reason the ACSI uses DKMs is to remove ambiguity when analysis methodologies are viewed by students who have never performed spacecraft verification by analysis. The cost of being wrong is too high, so a formal language is used to describe analysis methodology. Formal languages have a syntax, or how an expression is structured, and semantics, or what an expression means when it is structured that way. DKMs use a concept called a "Semantic Triple" to eliminate ambiguity that has a subject, predicate, and object. The subject is how an expression starts. The predicate has a verb that may be passive or active. The object is the recipient of the predicate. As an example, "Scalar Parameter 1 is an input to Equation 1".

| **Subject** | **Predicate** | **Object** |
|---|---|---|
| Scalar Parameter 1 | is an input to | Equation 1 |

The expression is supposed to read like a natural language or a common sentence. The subject Scalar Parameter 1 and object Equation 1 are instances of two categories, Scalar Parameter and Equation. An instance of Scalar Parameter might be “Sun Sensor Mass” whereas an instance of Equation might be “Newton’s Second Law”. If those were put together, the expression would be “Sun Sensor Mass is an input to Newton’s Second Law”. This seems straightforward, and it is. The categories of entities that can exist in DKMs are defined by an Ontology Relationship Map (ORM) for the Technical Analysis Domain. This ORM is shown in Figure 2 and is provided in ACSI onboarding packages.

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Scalar Parameter | • is compared to | • is compared to | • is compared to | • is varied by | | • is an input to | | • is an input to | |
| Array Parameter | • is compared to | • is compared to | • is compared to | • is varied by | | • is an input to | | • is an input to | |
| Matrix Parameter | • is compared to | • is compared to | • is compared to | • is varied by | | • is an input to | | • is an input to | |
| State | | | | | | | | | |
| Source | • provides | • provides | • provides | | | • provides | • provides | | |
| Equation | • calculates | • calculates | • calculates | | | | | | |
| Modeling Environment Entity | | | | • Is varied by | | | | • is an input to | |
| Modeling Environment Module | • models | • models | • models | | | | | | |
| Modeling Environment | | | | | | | | • Is comprised of | |
| | Scalar Parameter | Array Parameter | Matrix Parameter | State | Source | Equation | Modeling Environment Entity | Modeling Environment Module | Modeling Environment |

**Figure 2: Technical Analysis Domain Ontology Relationship Map**

Figure 2 is read from the left column (the subject) to the right relationship (the predicate) and then down to the bottom row (the object) to create a semantic triple. When DKMs are created using only the available categories and relationships, no ambiguity exists regarding the execution of a given analysis methodology. Appendix B has a detailed list of category descriptions and hex codes for category colors, which were selected to maximize visibility by colorblind individuals.

### 2.2.2 Knowledge Point Integration Diagrams

DKMs are created for all KPs, and KPs are strung together culminating in the calculation or simulation of a TPM. In the SAP, each TPM is given a figure showing which KPs culminate in a DKM. DKMs will be integrated into the ACSI web-based tools, currently at www.abexmission.org. All DKMs will move online and be dynamically traversable for all KPs within a TPM. An example of a “Knowledge Point Integration

Diagram" for a given TPM is shown in Figure 3 wherein KPs are strung together culminating in the calculation of the two TPMs on the far right. The diagram in Figure 3 is not a Domain Knowledge Map. It's an organization of DKMs used for logical and representative purposes. It does not follow the strict semantics of DKMs, and it is used in the Subsystem Analysis Plan to say, "In order to calculate or model this TPM, we need to calculate or model these Knowledge Points in this order first." These diagrams are read from left to right. Figure 3 represents the calculation or simulation methodology for two highly coupled TPMs, but commonly one diagram is made per TPM.


**Figure 3: Example Knowledge Point Integration Diagram (imperfect)**

### 2.2.3 DKM Example

DKMs are made in draw.io, or what is now diagrams.net. As long as the shapes, colors, and label rules are followed in accordance with the calculation methodology, the DKM will be correct. DKMs are difficult to make correctly, and the ACSI is in the process of developing a platform that allows for DKM creation in the correct format. Until that is ready, it is possible to make bad DKMs resulting in rework. Here are some guidelines for making useful DKMs.

- All KPs are ovals, everything else is a rectangle.
- If it has its own DKM, it's an oval KP
- Keep all shapes a similar size, but it does not have to be exact. Readability is most important.
- All lines must include a distinct relationship using a semantic triplet.
- If keeping relationships near each other on the lines looks bad, move the line before trying to convey that multiple lines have a single relationship prescribed to it.
- Make the DKMs read left-to-right, when possible, to follow the natural direction of reading.
    - There is an error in Figure 4 on this bullet, so watch out for this.
- The color of the entity or block must use established categories of the domain (e.g., scalar parameter, matrix parameter, etc.) from the Ontology Relationship Map
- Equations are numbered based on SAP sections. You'll need to define the equation numbers and SAP sections in tandem, so put X.X.X.X at first if you don't know the equation numbers.

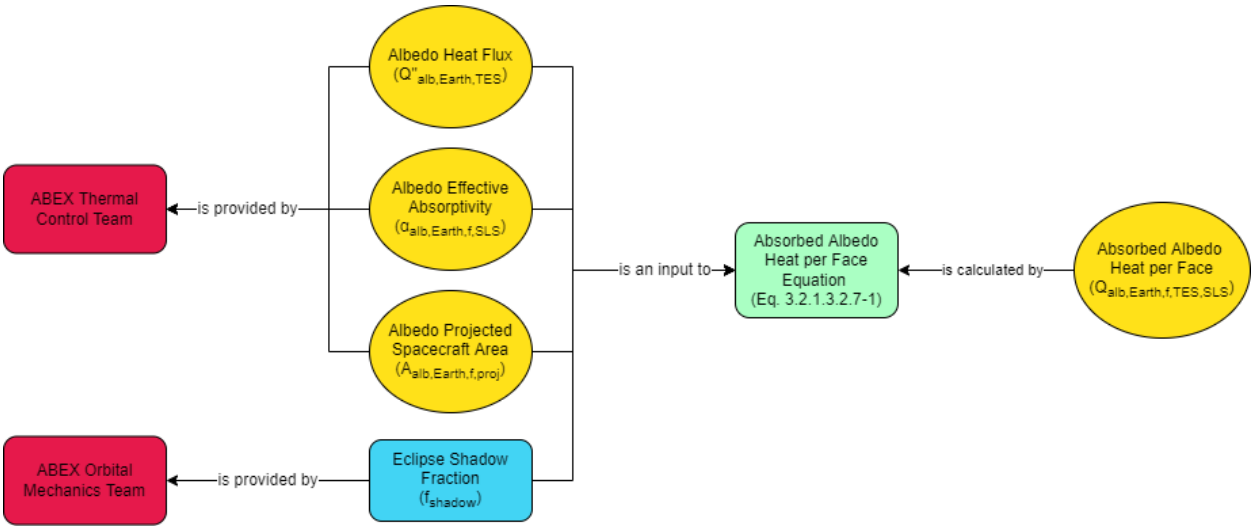


**Figure 4: Domain Knowledge Map Example**

In Figure 4 is an example of a DKM. It is read from left to right, starting with sources on the left. The sources state where information originates, and here it says three array parameter KPs are provided by the ABEX Thermal Control Team. That's the starting point to read DKMs. However, looking to Figure 2 the relationship from source to parameter is "provides" instead of "is provided by". So the arrows should be going from left to right and say "provides" rather than going from right-to-left saying "is provided by." The same error is made from the equation block to the final KP wherein it should say the equation "calculates" the array parameter instead of saying the array parameter "is calculated by" the equation. Little things like this contribute greatly to the readability of DKMs.

### 2.2.4 DKM Remarks

Remember that DKMs are visual representations of analysis methodology, and analysis is a type of verification activity. The list of TPMs, KPs, and parameters involved in an analysis are represented in the SAP along with all the DKMs. If a subsystem has a thorough SAP, it can be used as one of the best onboarding documents for students new to that subsystem.

# 3 Product Breakdown Structure

## 3.1 Purpose

The Product Breakdown Structure (PBS) organizes system elements hierarchically to plan integration work around hardware and software that a project needs to make or buy. The PBS for all ACSI missions will capture both flight and development products needed to mature system technology. The PBS will also establish a common terminology for the system for the team and for the project's systems engineers. Flight products are the hardware and software elements that will fly. Development products are the hardware and software elements needed to raise the subsystem Technology Readiness Level (TRL) but will not fly. The PBS is developed as a standalone document but ultimately exists in the subsystem Development and Integration Plan (DIP), which plans the activities needed to mature a subsystem to TRL 5. The PBS is also included in the subsystem Qualification Plan (QP), which plans the activities needed to mature a subsystem to TRL 6.

## 3.2 Guidance

When creating the PBS for a subsystem, some guidance is offered for consideration:

- Products are organized in a tree structure that represents how products fit within (or integrate) into each other in the final implementation.
    - Examples:
        - 1.1 would go under 1
        - 3.4.3.5 would go under 3.4.3
        - 7.1.1.1.2 would go under 7.1.1.1
    - Flight hardware for a given subsystem is X.1.
    - Prototype development hardware for a given subsystem is X.2
    - Additional development hardware for a given subsystem may be X.3 or X.4
        - X.3 may or may not be needed and is dependent on the subsystem design. It's subjective
    - The X's correspond to the below numbering scheme for a subsystem
- Each subsystem has its own subsystem ID for the first number
    - 1. Structures & Mechanisms
    - 2. Command & Data Handling
    - 3. Electrical Power System
    - 4. Guidance, Navigation, & Control
    - 5. Payload
    - 6. Telemetry, Tracking, & Command
    - 7. Thermal Control
    - 8. Flight Software
- The leveling approach should go as far as needed to capture the major elements of the project but does not need to recreate a full parts list (such as each capacitor and resistor).
    - When deciding whether to include a component, think in terms of integration. If it is a capacitor and resistor that will be purchased on a PCB, the PCB would be integrated, not the components on the PCB.
- Development is to be included in the PBS and should be organized around major technology advancement hardware units/setups. There may be multiple development levels.
    - This corresponds to X.2, X.3, or X.4+. X.2 is prototype hardware, which usually undergoes TRL 6 environmental testing. On fast schedule missions, X.2 will go up to TRL 5 and X.1 will be undergo "protoflight" TRL 6 testing.

An example of this is shown below in Table 3 for a fictional system.

**Table 3: Example Product Breakdown Structure for Structures & Mechanisms.**

| PBS ID | Product Name | Product Description |
|---|---|---|
| 1 | Structures & Mechanisms | System example name |
| 1.1 | Flight Hardware | Flight subsystem elements |
| 1.1.1 | Chassis | Load-bearing spacecraft structure |
| 1.1.1.1 | +Z Chassis Face | Spacecraft structural element |
| 1.1.1.2 | -Z Chassis Face | Spacecraft structural element |
| 1.1.1.3 | +Y Chassis Face | Spacecraft structural element |
| 1.1.1.4 | -Y Chassis Face | Spacecraft structural element |
| 1.1.1.5 | +X Chassis Face | Spacecraft structural element |
| 1.1.1.6 | -X Chassis Face | Spacecraft structural element |
| 1.1.2 | Fasteners | All fasteners on the satellite |
| 1.1.2.1 | Chassis Fasteners | Fasteners between structural elements |
| 1.1.2.1.1+ | | Add subgroups as needed |
| 1.1.2.2 | Harness Fasteners | Fasteners between harnesses, components, and structures |
| 1.1.2.2.1+ | | Add subgroups as needed |
| 1.1.2.3 | Avionics Fasteners | Fasteners between avionics, components, and structures |
| 1.1.2.3.1+ | | Add subgroups as needed |
| 1.1.3 | Harnesses | Component harnessing equipment |
| 1.1.3.1 | Component Harnesses | Structural harnesses for components |
| 1.1.3.1.1+ | | Add subgroups as needed |
| 1.1.3.2 | Cable Harnesses | Harnesses for cabling |
| 1.1.3.2.1+ | | Add subgroups as needed |
| 1.1.4 | Mechanisms | Any mechanism on the spacecraft with motion |
| 1.1.4.1 | Hold & Release Mechanisms | Hold the solar array wings down |
| 1.1.4.1.1 | H&RM 1 | Holds the +X solar array wing |
| 1.1.4.1.2 | H&RM 2 | Holds the –X solar array wing |
| 1.1.4.2 | Deployment Switches | Determines when the spacecraft has ejected from the dispenser |
| 1.1.4.2.1 | Deployment Switch 1 | First deployment switch on the –Z face close to +X face |
| 1.1.4.2.2 | Deployment Switch 2 | Second deployment switch on the –Z face close to –X face |
| 1.2 | Prototype Hardware | Prototype subsystem elements |
| 1.2.1 | Chassis | Load-bearing spacecraft structure |
| 1.2.1.1 | +Z Chassis Face | Spacecraft structural element |
| 1.2.1.2 | -Z Chassis Face | Spacecraft structural element |
| 1.2.1.3 | +Y Chassis Face | Spacecraft structural element |
| 1.2.1.4 | -Y Chassis Face | Spacecraft structural element |
| 1.2.1.5 | +X Chassis Face | Spacecraft structural element |
| 1.2.1.6 | -X Chassis Face | Spacecraft structural element |
| 1.2.2 | Fasteners | All fasteners on the satellite |
| 1.2.2.1 | Chassis Fasteners | Fasteners between structural elements |

| | | |
|---|---|---|
| 1.2.2.1.1+ | | Add subgroups as needed |
| 1.2.2.2 | Harness Fasteners | Fasteners between harnesses, components, and structures |
| 1.2.2.2.1+ | | Add subgroups as needed |
| 1.2.2.3 | Avionics Fasteners | Fasteners between avionics, components, and structures |
| 1.2.2.3.1+ | | Add subgroups as needed |
| 1.2.3 | Harnesses | Component harnessing equipment |
| 1.2.3.1 | Component Harnesses | Structural harnesses for components |
| 1.2.3.1.1+ | | Add subgroups as needed |
| 1.2.3.2 | Cable Harnesses | Harnesses for cabling |
| 1.2.3.2.1+ | | Add subgroups as needed |
| 1.2.4 | Mechanisms | Any mechanism on the spacecraft with motion |
| 1.2.4.1 | Hold & Release Mechanisms | Hold the solar array wings down |
| 1.2.4.1.1 | H&RM 1 | Holds the +X solar array wing |
| 1.2.4.1.2 | H&RM 2 | Holds the –X solar array wing |
| 1.2.4.2 | Deployment Switches | Determines when the spacecraft has ejected from the dispenser |
| 1.2.4.2.1 | Deployment Switch 1 | First deployment switch on the –Z face close to +X face |
| 1.2.4.2.2 | Deployment Switch 2 | Second deployment switch on the –Z face close to –X face |
| 1.3 | Development Hardware | |
| 1.3.1 | Development Chassis | |
| 1.3.1.1 | Dev +Z Chassis face | Fabricated to test CNC capabilities |
| 1.3.1.2 | Dev –Y Chassis face | Fabricated to test CNC capabilities |
| 1.3.1.3 | Dev +X Chassis face | Fabricated to test CNC capabilities |
| 1.3.2 | Development Harnesses | |
| 1.3.2.1 | Dev Component Harness | |
| 1.3.2.2 | Dev Cable Harness | |

## 3.3 PBS Remarks

Visible on Table 3 is that X.1, flight hardware, and X.2, prototype development hardware, are very similar if not identical. What that means is we don't anticipate the hardware changing much between prototype testing and flight. X.3, development hardware, is very different from prototype and flight hardware. Non-prototype development hardware might include early designs of a select few components, cheap processors such as raspberry pis to do small control operations, or sensors to measure parameters. Development hardware does not have to fly, so X.3 or X.4 is generally where products are smaller, targeted, or cheaper than the flight hardware. Anywehre it says "Add subgroups as needed" could be expanded into multiple lines, and you would not have a "+" at the end of the item number.

# 4 Integration & Testing Process

Integration and Testing (I&T) is a common grouping of important SE concepts. This grouping is intended to describe the gradual combination of less complex systems into a system of increasing complexity and the confirmation that the combination of less complex systems functions as anticipated. Testing is an implicit part of integration and an explicit method of verification. For a given subsystem design, engineers must verify component functionality, integrate components into subsystems, verify subsystem functionality, integrate subsystems into systems, and verify system functionality. The system is then put through qualification tests to raise its Technology Readiness Level. Verification by test at each individual step is the least risky option for component, subsystem, or system functionality verification, but each test costs both time and money. Programs have been cancelled because they tested too much and too often, but failure to plan sufficient integration testing can also led to mission failure. System integration planning must balance both cost and risk when developing an integration strategy. The following material provides insight into what I&T is and how to organize it effectively.

## 4.1 Integration

Integration is one of the most challenges aspects of engineering complex systems. A complicated system is one with many inputs and outputs; a complex system is one with many interfaces for mass, energy, data, or physical interactions. Issues or shortcomings with design specifications are exposed, and the entire process is increasingly difficult as the number of parts or interfaces grows. Integration is not the same as assembly. Assembly emphasizes the mechanical mating of components to form a system while integration encompasses the entire process of combining less complex function to achieve a system satisfying its requirements. Integration is accomplished by defining Integration Points (IP), Configuration Items (CI), and Integration Chains (IC).

### 4.1.1 Configuration Items

A CI is a combination of two or more components or subsystems. It's what happens when two things with individual functionality are combined. Two PCBs put into a stack might be a CI, or a heater applied to a battery box might be a CI. CIs are steppingstones on the way to the final system. If a system is put together all at once and doesn't work, it might be difficult to determine what went wrong or how to fix the problem. Functionality of a CI must be determined at each IP to move from one CI to the next; functionality must therefore be verified by inspection, demonstration, analysis, or test, usually test.

### 4.1.2 Integration Points

An IP happens when a CI is created; the IP is the event while the CI is the combination of components or subsystems.

### 4.1.3 Integration Chains

An IC is a planned set of IPs that result in the successful operation of some system functionality. A set of functions and subsystems that result in a critical parameter or TPM is called a chain. If the TPM includes computational operations or code execution, this chain may include real-time, information intensive interfaces or the inclusion of sensors, actuators, or mechanisms. While it may seem easier to start with the least complex parts and work to combine components to create a more complex system, it is often more efficient to start with the finalized system or subsystem and work backwards to define an IC.

### 4.1.4 Integration Considerations

While ICs are determined by working backwards from a complete system, integration is enacted first by the provisional testing of individual components at the lowest levels. Faults can be identified, isolated, and recovered from within a small and controllable scope, and problems caught at lower levels are easier and cheaper to fix. While not every miniscule combination of components merits testing, for this reason testing

at lower levels should not be avoided. When deciding if a combination of components represents a CI that must be tested for functionality, several questions can be asked.

- How critical is the component or combination of the components to the operation of the subsystem?
- What aspects of the CI are sufficiently close to final operation that feedback from this CI makes sense?
- How much time, money, or effort must be invested in this IP?

### 4.1.5 Software Integration

Hardware integration is moderately linear; there are direct paths joining other paths that result in a final product. Software integration, conversely, is modular and incremental. When integrating software into hardware, there are four levels of decreasing abstraction: Model-In-The-Loop (MIL), Software-In-The-Loop (SIL), Processor-In-The-Loop (PIL), and Hardware-In-The-Loop (HIL). As the software fidelity increases, the more realistic the test cases become. These XIL tests can be planned into an IC. In Table 4, simulated refers to an entirely computational representation without hardware, development signifies an intermediate or prototype design or deployment, and production means the flight model.

**Table 4: XIL Nomenclature descriptions and connection to major integration elements.**

| Elements | Model-In-The-Loop | Software-In-The-Loop | Processor-In-The-Loop | Hardware-In-The-Loop |
|---|---|---|---|---|
| **Code** | Development | Development | Production | Production |
| **Controller** | Simulated | Development | Production | Production |
| **Sensors** | Simulated | Simulated | Simulated | Production |
| **Environment** | Simulated | Simulated | Simulated | Simulated |

## 4.2 Testing

Testing means operating a system (or one or more of its parts) in a predefined way to verify its behavior. It is one of the four methods of verification.

### 4.2.1 Testing Types

When planning to test individual components, subsystems, systems, or CIs, each test includes both a test type and test category. Development tests are generally either functional tests or integration tests. Qualification tests are environmental tests, and very often environmental testing includes and builds upon functional testing. Acceptance tests are a type of environmental test that consider workmanship or final system checkout rather than testing the system to its limits of operation within a given operational environment.

#### 4.2.1.1 Type 1: Development Tests

Development tests validate new design concepts, techniques, configurations, or the combination of any of those. Development testing occurs early during the development life cycle phase and typically confirm aspects of a design such as performance margins, manufacturability, reliability, failure modes, or a systems ability to be tested. Documentation for development tests may not be as rigid as that of qualification or acceptance tests because development tests occur earlier and more frequently in the design life cycle. Operational conditions should be varied during development tests. The purpose is to inform system design (Larson, Kirkpatrick, Sellers, Thomas, & Verma, 2018).

#### 4.2.1.2 Type II: Qualification Tests

Qualification tests validate a design for the intended operational environment. While the test should occur for the extremes of the environment, the objective is to show evidence of applicable design margins, not to

cause failure modes unrealistic to nominal operation. Qualification tests are rigorous and usually expensive, so they should be performed when both a hardware and software design are finalized. Qualification testing occurs directly before acceptance testing of build products.

#### 4.2.1.3 Type III: Acceptance Tests

Acceptance testing assures conformance to specification requirements and provides quality control assurance against workmanship or material deficiencies (Larson, Kirkpatrick, Sellers, Thomas, & Verma, 2018). Acceptance tests verify requirements associated with build-to or product specifications, readiness for delivery to customers, and acceptance by the customers. They check for workmanship, not the design of the system. The purpose for acceptance tests is to show that the production version of a system was manufactured, assembled, and integrated correctly. These testing activities occur for each item produced whereas qualification tests occur only once for a given design. Acceptance tests occur in the late development life cycle phase.

### 4.2.2 Test Categories

Test categories exist within test types. Both the type and category should be specified for an IP.

#### 4.2.2.1 Functional Tests

Functional tests answer the question, “Does it work?” They are used to verify the system’s functional requirements at all levels, be those component, subsystem, or the spacecraft level. Deployment of a solar array, antenna frequency tests, or attitude control system operation would be functional tests. It is important to note the difference between function and capability. A capability is a function at a level of performance; a function itself is unitless. Testing for a capability counts as testing for a function, but the distinction should be made in test planning whether the test is for a function or a function at a specified level of performance. (Larson, Kirkpatrick, Sellers, Thomas, & Verma, 2018).

#### 4.2.2.2 Integration Tests

Integration tests answer the question, “Do the pieces work together as intended?” At a given IP, a CI may be tested using both integration and functional tests. Integration tests may include mechanical, electrical, or software checks, with software requiring the most planning. Electrical integration checks may include data or power interfaces, power consumption, data transfer, grounding, or soldering. Software integration checks may include any version of XIL testing. Mechanical interfaces, alignments, and mass properties should be checked during integration (Larson, Kirkpatrick, Sellers, Thomas, & Verma, 2018).

#### 4.2.2.3 Environmental Tests

Environmental tests answer the question, “Does it work in the operational environment?” Note that the environmental test question includes and builds upon the functional test question. Environmental tests verify that components will withstand conditions the system will be exposed to and typically include thermal vacuum, mechanical vibration, mechanical shock, electromagnetic compatibility, or radiation testing. Some systems may require humidity, salt spray, or leakage testing (Larson, Kirkpatrick, Sellers, Thomas, & Verma, 2018).

## 4.3 Documenting Integration Flows

To describe the integration flow, an integration chain of test activities can be represented as a UML diagram or similar style diagram on draw.io or diagrams.net. The ACSI is working on a tool to represent these cleanly as well. These diagrams appear in the last section of the Development and Integration Plan (DIP) and correspond with the preceding Product Breakdown Structure (PBS). The diagram is organized using horizontal swimlanes for each PBS item. Blocks with connecting lines represent tests, which may be development, qualification, or acceptance tests. Block colors indicate if they are function, integration, or environmental tests.

Each time one or more swimlanes, or PBS products, combine, that represents an integration point that usually calls for an integration test. Function tests are performed on a single product in a single lane, and most products do not need function tests before going to an integration test. Remember the integration test question, “Do the pieces work together as intended?” includes the question, “Does it work?” The IDs in the swimlanes should match the IDs in the PBS.

Tests that absolutely must show up in a subsystem integration flow are the TRL advancement tests, specifically TRL 4-6. The test for TRL 4 is an integration test in lab conditions, so it would be yellow and named with TRL 4 in the name. The test for TRL 5 is a development environmental test, so it would be red and also named explicitly. The test for TRL 6 is also an environmental test, but it’s a qualification environmental test. The flow from TRL 3-5 goes in the DIP, and the flow from TRL 5-6 goes in the QP.

Defining integration chains is a subjective task. Subsystem teams decide which PBS items need function tests or which grouping of PBS items can go directly to an integration test. Integration tests must always precede environmental tests.

For integration tests not explicitly organized along the subsystem PBS, such as a software integration test, a block can be placed between swimlanes to show its integration of them. An example of this is shown in Figure 5, with the color coding and naming scheme described in Table 5.

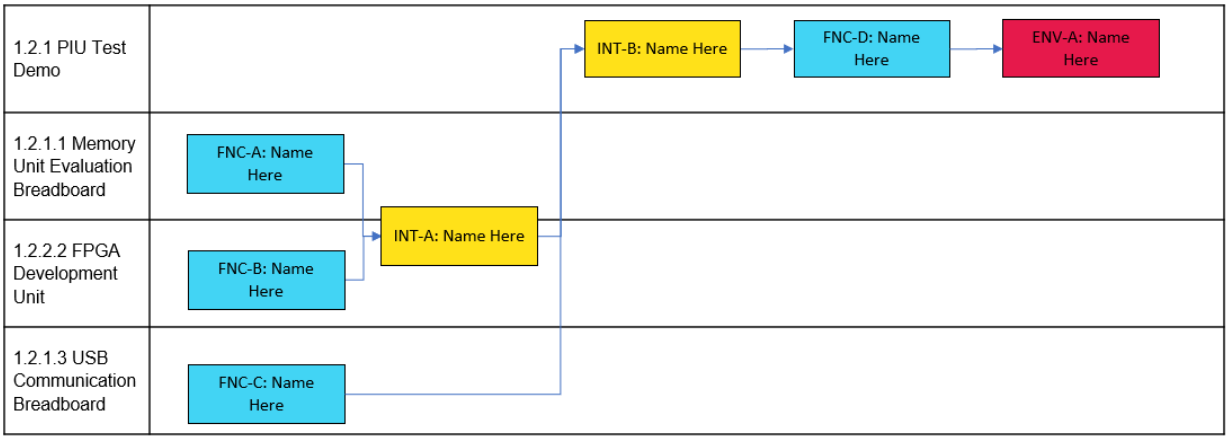


***Figure 5: An example of a test flow. Note that the INT-A test is not part of the PBS but a possible integration point to get towards PBS 1.2.1 and lives between the swimlanes it needs.***

**Table 5: Color codes for test types.**

| Test Type | Acronym / Shorthand | Color Code |
|---|---|---|
| **Functional Test** | FNC-ID# | *Hex#: 42D4F4* |
| **Integration Test** | INT-ID# | *Hex#: FFE119* |
| **Environmental Test** | ENV-ID# | *Hex#: E6194B* |

Note that in swimlane 1.2.1 PIU Test Demo, it shows an integration test, then a function test, then an environmental test. Generally, you would not need a function test in between an integration and environmental test.

# 5 Technology Readiness Levels

## 5.1 Purpose

Technology readiness is used to determine how ready a technology is to operate as a part of a system, based on its demonstrated functional ability within the target environment and how ready it is to be integrated within the flight system. Levels are defined within this to act as milestones during technology development, integration, and testing.

Importantly, not all development and qualification activities are TRL advancement. For example, assume a team is developing a new Printed Circuit Board (PCB) design using circuitry and components that have all flown on prior missions, but they have never been organized in this way prior to this new design. When that team builds and tests that PCB, they are not doing TRL advancement. That's just design and engineering. The team still needs to perform integration tests on a breadboard to hit what would be TRL 4, perform medium-fidelity environmental tests on a brassboard to get to TRL 5, and perform high-fidelity environmental tests on a prototype to get to TRL 6, but it is not "Technology Readiness Advancement" per se. At that point they are making sure the parts do not fall off the PCB when they shake it and bake it, but the system components are at whatever TRL they were based on previous flight experience the whole time.

## 5.2 Level Classifications

NASA defines 9 levels of technology readiness. These are described in Table 6 including the descriptions for what this means for the hardware and software elements of a technology with exit criteria expectation.

**Table 6: TRL definitions adapted from NASA NPR 7123.1C.**

| Lvl | Definition | Hardware Description | Software Description | Exit Criteria |
|---|---|---|---|---|
| **1** | Basic principles observed and reported. | Scientific knowledge generated underpinning hardware technology concepts/applications. | Scientific knowledge generated underpinning basic properties of software architecture and mathematical formulation. | Peer reviewed publication of research underlying the proposed concept/application. |
| **2** | Technology concept and/or application formulated. | Invention begins, practical application is identified but is speculative, no experimental proof or detailed analysis is available to support the conjecture. | Practical application is identified but is speculative, no experimental proof or detailed analysis is available to support the conjecture. Basic properties of algorithms, representations and concepts defined. Basic principles coded. Experiments performed with synthetic data. | Documented description of the application/concept that addresses feasibility and benefit. |
| **3** | Analytical and experimental critical function and/or characteristic proof of concept. | Analytical studies place the technology in an appropriate context and laboratory demonstrations, modeling and simulation validate analytical prediction. | Development of limited functionality to validate critical properties and predictions using non-integrated software components. | Documented analytical/experi-mental results validating predictions of key parameters. |
| **4** | Component and/or breadboard validation in | A low fidelity system/component breadboard is built and operated to demonstrate basic functionality and | Key, functionally critical, software components are integrated, and functionally validated, to establish interoperability and begin architecture development. | Documented test performance demonstrating agreement with analytical |

| Lvl | Definition | Hardware Description | Software Description | Exit Criteria |
|---|---|---|---|---|
| | laboratory environment. | critical test environments, and associated performance predictions are defined relative to the final operating environment. | Relevant Environments defined and performance in this environment predicted. | predictions. Documented definition of relevant environment. |
| **5** | Component and/or breadboard validation in relevant environment. | A medium fidelity system/component brassboard is built and operated to demonstrate overall performance in a simulated operational environment with realistic support elements that demonstrates overall performance in critical areas. Performance predictions are made for subsequent development phases. | End-to-end software elements implemented and interfaced with existing systems/simulations conforming to target environment. End-to-end software system, tested in relevant environment, meeting predicted performance. Operational environment performance predicted. Prototype implementations developed. | Documented test performance demonstrating agreement with analytical predictions. Documented definition of scaling requirements. |
| **6** | System/sub-system model or prototype demonstration in an operational environment. | A high-fidelity system/component prototype that adequately addresses all critical scaling issues is built and operated in a relevant environment to demonstrate operations under critical environmental conditions. | Prototype implementations of the software demonstrated on full-scale realistic problems. Partially integrate with existing hardware/software systems. Limited documentation available. Engineering feasibility fully demonstrated. | Documented test performance demonstrating agreement with analytical predictions. |
| **7** | System prototype demonstration in an operational environment. | A high-fidelity engineering unit that adequately addresses all critical scaling issues is built and operated in a relevant environment to demonstrate performance in the actual operational environment and platform (ground, airborne, or space). | Prototype software exists having all key functionality available for demonstration and test. Well integrated with operational hardware/software systems demonstrating operational feasibility. Most software bugs removed. Limited documentation available. | Documented test performance demonstrating agreement with analytical predictions. |
| **8** | Actual system completed and "flight qualified" through test and demonstration. | The final product in its final configuration is successfully demonstrated through test and analysis for its intended operational environment and platform (ground, airborne, or space). | All software has been thoroughly debugged and fully integrated with all operational hardware and software systems. All user documentation, training documentation, and maintenance documentation completed. All functionality successfully demonstrated in simulated operational scenarios. Verification and Validation (V&V) completed. | Documented test performance verifying analytical predictions. |

| Lvl | Definition | Hardware Description | Software Description | Exit Criteria |
|---|---|---|---|---|
| **9** | Actual system flight proven through successful mission operations. | The final product is successfully operated in an actual mission. | All software has been thoroughly debugged and fully integrated with all operational hardware/software systems. All documentation has been completed. Sustaining software engineering support is in place. System has been successfully operated in the operational environment | Documented mission operational results. |

## 5.3 Raising TRL

In general, you can conceptualize the process of raising a TRL (from a subsystem perspective) in the following manner:

1. Define the requirements for a subsystem. For ACSI missions, these are provided by Management.
2. Define an architecture (what) that characterizes the functions described in the requirements.
3. Determine the TPMs that a design can provide to meet the specification requirements.
4. Define a design (how) that specifically details what components or subsystems are providing the functionality described in the requirements. Characterize the entire design space to meet the problem space before selecting the design.
5. Break the subsystem design into constituent components, organized into a PBS. For ACSI missions we include both the development hardware elements and flight hardware.
6. Establish the current TRL of the subsystem based on the concept design.
7. Determine the entry and exit criteria for each TRL from its current point to TRL 6. Determine the test categories required to verify compliance of the subsystem design to its requirements as they relate to each TRL. Define the development hardware we need and include that into the PBS.
8. Determine the configuration items that could exist and associated integration chains for each TRL advancement. This might include several TRL steps at once.
9. Once the subsystem has demonstrated TRL 6 from tests done independently to it, the flight unit is ready to be built and tested to the operational environment with the whole spacecraft together to achieve TRL 8.
10. After flight the subsystem is at TRL 9.

## References


[1] "NASA Systems Engineering Handbook," p. 297, 2016.

[2] C. S. Wasson, Systems Engineering Analysis, Design, and Development, John Wiley & Sons, 2016.

[3] W. J. Larson, D. Kirkpatrick, J. J. Sellers, L. D. Thomas and D. Verma, Applied Space Systems Engineering, McGraw Hill, 2018.

# Appendix A Acronyms, Terminology, & Nomenclature

## A.1 Acronyms

| Acronym | Definition |
|---|---|
| **ABEX** | Alabama Burst Energetics Explorer |
| **ACSI** | Alabama CubeSat Initiative |
| **ASGC** | Alabama Space Grant Consortium |
| **CAD** | Computer-Aided Design |
| **C&DH** | Command & Data Handling |
| **CDR** | Critical Design Review |
| **CE** | Chief Engineer |
| **CI** | Configuration Item |
| **CLB** | Configurable Logic Block |
| **COTS** | Commercial Off The Shelf |
| **CS** | Chief Scientist |
| **CSLI** | CubeSat Launch Initiative |
| **DAC** | Design Analysis Cycle |
| **DIP** | Development and Integration Plan |
| **DKM** | Domain Knowledge Map |
| **EAR** | Export Administration Regulations |
| **EPMs** | Educational Performance Measures |
| **EPS** | Electrical Power System |
| **FPGA** | Field Programmable Gate Array |
| **FSW** | Flight Software |
| **GN&C** | Guidance, Navigation, & Control |
| **GPS** | Global Positioning System |
| **HV** | High Voltage |
| **IC** | Integration Chain |
| **ICP** | Instrument Calibration Plan |
| **IMS** | Integrated Master Schedule |
| **IMU** | Inertial Measurement Unit |
| **IP** | Integration Point |
| **ISM** | Integrated Systems Model |
| **ITAR** | International Traffic in Arms Regulations |
| **IV&T** | Integration, Verification, & Test |
| **KDP** | Key Decision Point |
| **KP** | Knowledge Point |
| **KPP** | Key Performance Parameters |
| **LSE** | Lead System Engineer |
| **MBSE** | Models Based System Engineering |
| **MCR** | Mission Concept Review |

| Acronym | Definition |
|---|---|
| **NDA** | Non-Disclosure Agreement |
| **NIST** | National Institute of Standards and Technology |
| **ORM** | Ontology Relationship Map |
| **PBS** | Product Breakdown Structure |
| **PC** | Program Coordinator |
| **PCB** | Printed Circuit Board |
| **PDR** | Preliminary Design Review |
| **PIU** | Payload Interface Unit |
| **PM** | Project Manager |
| **POP** | Period of Performance |
| **QP** | Qualification Plan |
| **QR** | Qualification Report |
| **QPSK** | Quadrature Phase Shift Keying |
| **SAP** | Subsystem Analysis Plan |
| **SAR** | Subsystem Analysis Report |
| **SE** | Systems Engineering |
| **SEGD** | Systems Engineering Guidance Document |
| **SEMP** | System Engineering Management Plan |
| **SIS** | Software Interface Specification |
| **SME** | Subject Matter Expert |
| **SRD** | System Requirements Document |
| **SRR** | System Requirements Review |
| **STP** | Subsystem Test Plan |
| **STR** | Subsystem Test Report |
| **TCP** | Technology Control Plan |
| **TID** | Total Ionizing Dose |
| **TPM** | Technical Performance Measure |
| **TQP** | Thermal Qualification Plan |
| **TQR** | Thermal Qualification Report |
| **TRL** | Technology Readiness Level |
| **TT&C** | Telemetry, Tracking, & Command |
| **V&V** | Verification and Validation |
| **WBS** | Work Breakdown Structure |

## A.2 Terminology

| Term | Description |
|---|---|
| **Acceptance** | A type of verification procedure specifically for testing and analysis. Acceptance test/analysis criteria show that the manufacturing/workmanship of the unit conforms to the design that was previously verified/qualified. Acceptance |

| Term | Description |
|---|---|
| | activities are performed on each of the flight units as they are manufactured and readied for flight/use (NASA Systems Engineering Handbook, 2016) |
| **Analysis** | Verification by analysis is a predicted compliance to requirements. The use of mathematical modeling and analytical techniques to predict the suitability of a design to stakeholder expectations based on calculated data or data derived from lower system structure end product verifications. Analysis is generally used when a prototype; engineering model; or fabricated, assembled, and integrated product is not available. Analysis includes the use of modeling and simulation as analytical tools (NASA Systems Engineering Handbook, 2016). |
| **Assembly** | The mechanical mating of components to form a system. |
| **Certification** | The audit process by which the body of evidence that results from the verification activities and other activities are provided to the appropriate certifying authority to indicate the design is certified for flight/use. The Certification activity is performed once regardless of how many flight units may be generated (NASA Systems Engineering Handbook, 2016). |
| **Configuration Item** | The combination of two components, subsystems, or systems of lesser complexity resulting in a combined assembly, subsystem, or system with greater complexity. Configuration Items exist at Integration Points; a sequence of Configuration Items along several Integration Points comprises an Integration Chain. |
| **Demonstration** | Verification by demonstration is an observed compliance to requirements accomplished by showing that the use of an end product achieves the individual specified requirement. It is generally a basic confirmation of performance capability, differentiated from testing by the lack of detailed data gathering. Demonstrations can involve the use of physical models or mock-ups (NASA Systems Engineering Handbook, 2016). |
| **Inspection** | Verification by inspection is a documented compliance to requirements. The visual examination of a realized end product. Inspection is generally used to verify physical design features or specific manufacturer identification (NASA Systems Engineering Handbook, 2016). |
| **Integration** | The process of combining less complex functions, understanding those functions, and controlling those functions to achieve a system satisfying its requirements. |
| **Integration Chain** | A series of Integration Points. Integration Chains can be represented as tree or fishbone diagrams where many components, subsystems, or systems of lesser complexity are combined as Configuration Items at Integration Points to create a system of higher complexity. Integration Chains are generally defined to realize a Technical Performance Measure. |
| **Integration Point** | The location on a schedule where two or more components, subsystems, or systems of lesser complexity are combined as a Configuration Item with greater complexity. A series of Integration Points comprises an Integration Chain. |
| **Interface** | An interface represents a constraint based on the logical and physical boundary conditions between two or more entities within a level of abstraction, between System of Interest elements, between other mission systems, between enabling systems, or between the System of Interest and its Operational Environment. Interfaces can be for physical connection, energy transfer (power or heat), matter, or data (Wasson, 2016). |
| **Key Performance Parameter** | Those capabilities or characteristics (typically engineering-based or related to health and safety or operational performance) considered most essential for successful mission accomplishment. They characterize the major drivers of |

| Term | Description |
|---|---|
| | operational performance, supportability, and interoperability (NASA Systems Engineering Handbook, 2016). |
| **Knowledge Point** | Any parameter needed to understand a system's environment, behavior, structure, or operations that can be calculated or modeled. Non-TPM KPs are subjectively determined from a total parameter list. |
| **Mode** | An abstract configuration, condition, or process that occurs with or without a corresponding physical state in a component, subsystem, or system at a given time. A non-tangible, non-physical concept. |
| **Model** | A mathematical representation of reality (NASA Systems Engineering Handbook, 2016). |
| **Operational Environment** | The surrounding systems, materials, or occurrences defining a system's ability to externally interact. The Operational Environment is comprised of a Human Systems Environment, a Natural Environment, and an Induced Environment (Wasson, 2016). |
| **Qualification** | A subset of the verification program that is performed at the extremes of the environmental envelope and will ensure the design will operate properly with the expected margins. Qualification is performed once regardless of how many flight units may be generated as long as the design doesn't change. |
| **Simulation** | The manipulation of a model (NASA Systems Engineering Handbook, 2016). |
| **State** | A physical mechanical configuration, environmental condition, operational condition, or other physical condition that either happens to or is initiated by a component, subsystem, or system at a given time. |
| **Technical Performance Measure** | A set of performance measures that are monitored by comparing the current actual achievement of the parameters with that anticipated at the current time and on future dates (NASA Systems Engineering Handbook, 2016) |
| **Test** | Verification by test is a measured compliance to requirements. : The use of an end product to obtain detailed data needed to verify performance or provide sufficient information to verify performance through further analysis. Testing can be conducted on final end products, breadboards, brassboards, or prototypes. Testing produces data at discrete points for each specified requirement under controlled conditions and is the most resource-intensive verification technique. As the saying goes, "Test as you fly, and fly as you test" (NASA Systems Engineering Handbook, 2016). |
| **Validation** | Validation of a product shows that the product accomplishes the intended purpose in the intended environment—that it meets the expectations of the customer and other stakeholders as shown through performance of a test, analysis, inspection, or demonstration (NASA Systems Engineering Handbook, 2016). |
| **Verification** | Verification is a formal process, using the method of test, analysis, inspection or demonstration, to confirm that a system and its associated hardware and software components satisfy all specified requirements. The Verification program is performed once regardless of how many flight units may be generated as long as the design doesn't change (NASA Systems Engineering Handbook, 2016). |

## A.3 Nomenclature

| Symbol | Description | Unit |
|---|---|---|
| | | |

| | | |
|---|---|---|
| | | |
| | | |
| | | |

# Appendix B Technical Analysis Domain Categories

An organization of Technical Analysis Domain entities is found in *Table 7*.

**Table 7: Technical Analysis Domain Entities**

| Categories | Descriptions | Hex # |
|---|---|---|
| **Scalar Parameter** | This is a (1x1) scalar value including all rational numbers. All parameters must either be calculated, modeled, or sourced on a DKM. | *1ECAF2* |
| **Array Parameter** | This is a set of (1xN) scalar values including all rational numbers. Arrays can be captured as data, provided by components, or created from a set of scalars. All arrays must have their 1xN values listed in the variable subscript. | *FFE119* |
| **Matrix Parameter** | This is a set of (MxN) scalar values including all rational numbers. Matrices might be generated by components or used as parts of a design scheme. Matrices can be specific to the defined analysis method. A stress or temperature distribution output from FEA would be a matrix. All matrices must have their MxN values listed in the variable subscript. | *B560BC* |
| **State** | A physical mechanical configuration, environmental condition, operational condition, or other physical condition that either happens to or is initiated by a component, subsystem, or system at a given time. | *A9A9A9* |
| **Source** | Sources are either other satellite subsystem teams or source material. Sources provide parameters. | *E6194B* |
| **Equation** | Equations are used to calculate parameters. Equations generally have more than one parameter that “is an input to” the equation. It is good practice to provide sources for equations where possible. | *79FF9F* |
| **Modeling Environment Entity** | A Modeling Environment Entity is something that only exists in the context of a Modeling Environment (e.g., mesh density for FEA or a CAD model). | *D6428F* |
| **Modeling Environment Module** | Modeling Environment Modules encapsulate any computational modeling scheme designed to output a parameter for analysis purposes. Modeling Environments can have multiple Modeling Environment Modules. MATLAB functions or subroutines are Modeling Environment Modules. FEA platforms like ABAQUS have vibrational and meshing modules that serve as distinct Modeling Environment Modules. Data Visualization can also be considered a module. | *469990* |
| **Modeling Environment** | A Modeling Environment is the platform or program an engineer, architect, or designer model something in (e.g., MATLAB, ABAQUS, Thermal Desktop, Simulink, etc.). The organized creation, development, and execution of models is performed within a Modeling Environment Module that exists within a Modeling Environment. | *F58231* |

## B.1 Scalar Parameter

This is a (1x1) scalar value including all rational numbers. **All parameters must either be calculated, modeled, or sourced on a DKM.**

Potential Relationships:

- Is compared to Scalar Parameter
- Is compared to Array Parameter
- Is compared to Matrix Parameter
- Is varied by State

- Is an input to Equation
- Is an input to Modeling Environment Module

## B.2 Array Parameter

This is a set of (1xN) scalar values including all rational numbers. All arrays must have their 1xN values listed in the variable subscript.

Potential Relationships:

- Is compared to Scalar Parameter
- Is compared to Array Parameter
- Is compared to Matrix Parameter
- Is varied by State
- Is an input to Equation
- Is an input to Modeling Environment Module

## B.3 Matrix Parameter

This is a set of (MxN) scalar values including all rational numbers. All matrices must have their MxN values listed in the variable subscript.

Potential Relationships:

- Is compared to Scalar Parameter
- Is compared to Array Parameter
- Is compared to Matrix Parameter
- Is varied by State
- Is an input to Equation
- Is an input to Modeling Environment Module

## B.4 State

The word “state” is often used interchangeably with “mode”, but the ACSI adopts a more specific definition.

State: A physical mechanical configuration, environmental condition, operational condition, or other physical condition that either happens to or is initiated by a component, subsystem, or system at a given time.

Mode: An abstract configuration, condition, or process that occurs with or without a corresponding physical state in a component, subsystem, or system at a given time. A non-tangible, non-physical concept.

In short, states are physical whereas modes are abstract. Solar arrays being stowed or deployed is a state. Antennas being powered down or transmitting is a state. Software operational categories are modes of operation. This can be confusing because state machines are often used to describe both modes and states.

States do not directly impose relationships as the subject of an ontological triple, but it is the object of several. As an example, spacecraft are often analyzed in their hottest and coldest conditions, which the ACSI refers to a Thermal Environment State. Degradation occurs in space due to radiation, so some parameters are varied by Beginning of Life (BOL) versus End of Life (EOL), which the ACSI refers to as Spacecraft Life State. An Array Parameter is varied by a State such as, “Transistor Bias Voltage is varied by Spacecraft Life State”. State relationships often appear vertically on DKMs.

## B.5 Source

Sources are either other subsystem teams or source material. Sources provide parameters, and all KPs used as inputs to equations or models on DKMs must be sourced from the team that calculated those KPs.

Potential Relationships:

- Provides Scalar Parameter
- Provides Array Parameter
- Provides Matrix Parameter
- Provides Equation
- Provides Modeling Environment Entity

## B.6 Equation

Equations are used to calculate parameters. Equations generally have more than one parameter that "is an input to" the equation. It is good practice to provide sources for equations where possible.

Potential Relationships:

- Calculates Scalar Parameter
- Calculates Array Parameter
- Calculates Matrix Parameter

## B.7 Modeling Environment Entity

A Modeling Environment Entity is something that only exists in the context of a Modeling Environment. When doing Finite Element Analysis (FEA), the mesh density is a concept that doesn't exist outside the context of FEA. It is still an input to the Modeling Environment Module, but it doesn't exist outside of the model. CAD is an example of a Modeling Environment Entity

Potential Relationships:

- Is an input to Modeling Environment Module
- Is varied by State

## B.8 Modeling Environment Module

Modeling Environment Modules encapsulate any computational modeling scheme designed to output a parameter for analysis purposes; Modeling Environments can have multiple Modeling Environment Modules. **Parameters are either calculated by an Equation, modeled by a Modeling Environment Module, or provided by a Source.** MATLAB functions or subroutines are Modeling Environment Modules. FEA platforms like ABAQUS have vibrational and meshing modules that serve as distinct Modeling Environment Modules.

Potential Relationships:

- Models Scalar Parameter
- Models Array Parameter
- Models Matrix Parameter

## B.9 Modeling Environment

A Modeling Environment is the platform or program an engineer, architect, or designer models something in. MATLAB, ABAQUS, Thermal Desktop, Simulink, and STK are all Modeling Environments. The organized creation, development, and execution of models is performed within a Modeling Environment Module that exists within a Modeling Environment.

Potential Relationships:

- Is comprised of Modeling Environment Module

## B.10 Potential Category Additions

The described categories are not the only categories that could exist within the Technical Analysis Domain ontology and may not comprise the categories necessary for robust description of material within a subdomain. C&DH and EPS, for example, might desire a category for complex numbers. TT&C might desire a category for strings. If it is deemed that the defined categories are insufficient to create a complete DKM for a TPM, notify ACSI Management team to get your category formally included.