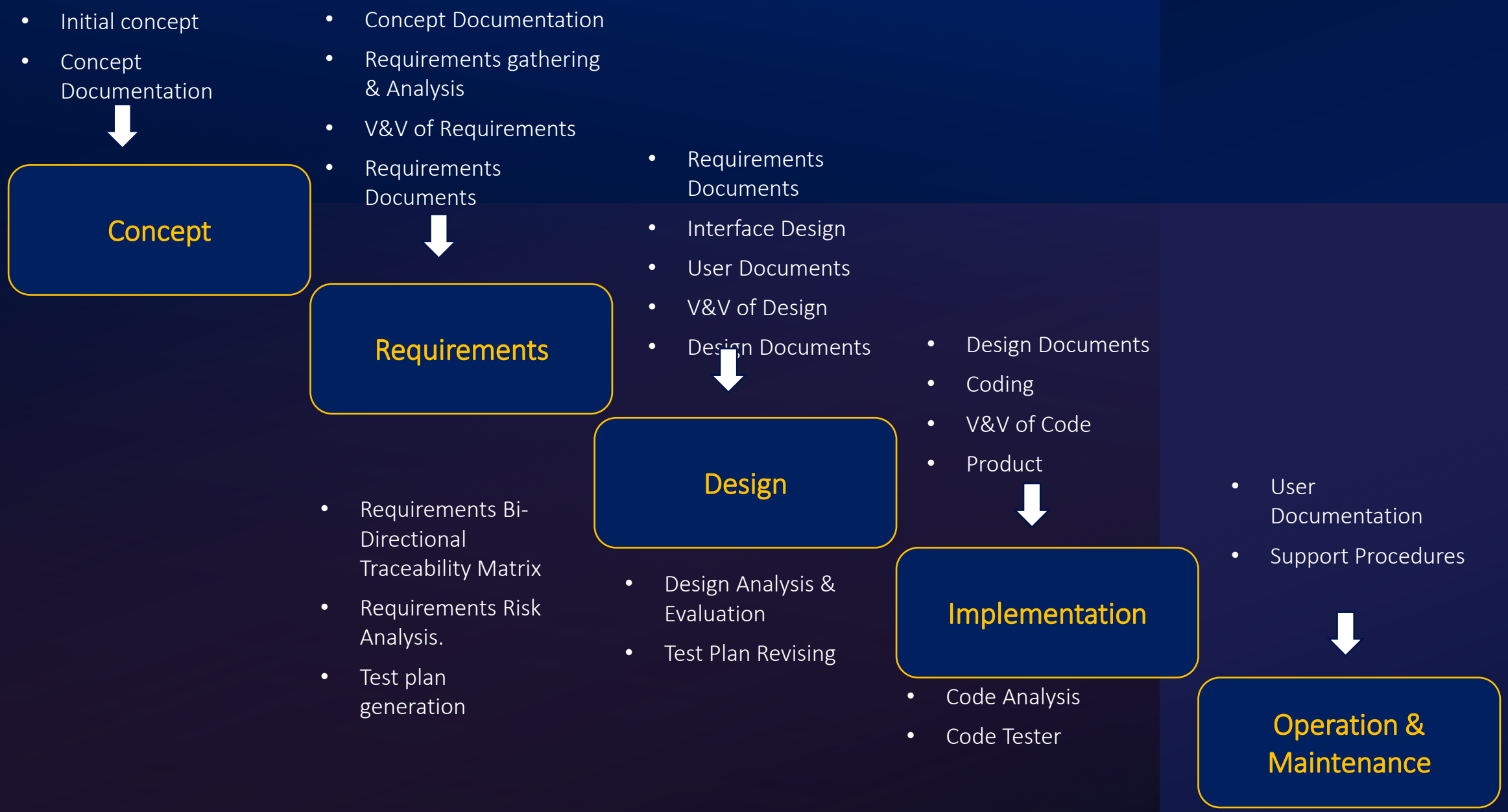
Systems Engineering for DoD

Lesson 5

Architecture & Design

Case 4

- Twice during the cold war nuclear war was almost launched due to software glitches. The first occurred at 2:25 am on June 3, 1980, systems at NORAD (United States) showed various and changing numbers of inbound missiles. This led to the launch of the airborne command post and putting the US nuclear systems on high alert. The cause was tracked down to a single faulty chip that was failing in a random fashion.
- The second incident occurred September 26, 1983. The early warning system for the Soviet Union twice reported the launch of American ICBMs. Fortunately an officer with the Soviet Air Defense Force realized these where false alarms. The error was caused by the systems satellite software not accounting for a rare alignment with high altitude clouds causing a glare.

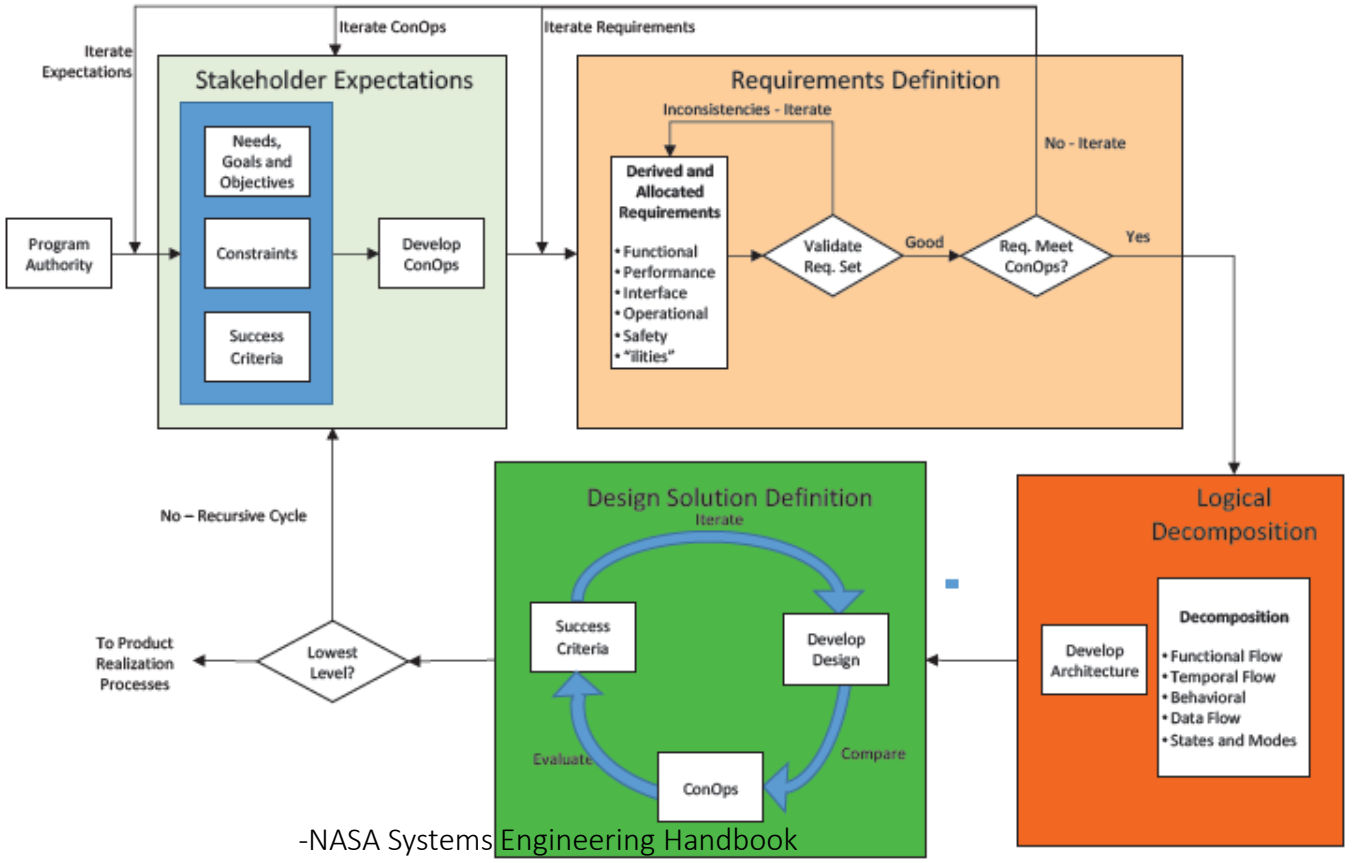

Initial concept
Concept Documentation
Concept
Concept Documentation
Requirements gathering & Analysis
V&V of Requirements
Requirements Documents
Requirements
Requirements Documents
Interface Design
User Documents
V&V of Design
Design Documents
Design
Design Documents
Coding
V&V of Code
Product
Implementation
User Documentation
Support Procedures
Operation & Maintenance
Requirements Bi-Directional Traceability Matrix
Requirements Risk Analysis.
Test plan generation
Design Analysis & Evaluation
Test Plan Revising
Code Analysis
Code Tester

Engineering – Lifecycle

NIST SP 800-60 addresses the role of security systems engineering within the lifecycle of U.S. Government owned systems. This same lifecycle should be applied to developing security in any environment. Thus, when implementing a new intrusion detection system, or in implementing new network policies, one should follow the ISO 15288 system development lifecycle (SEBok, 2018). That standard includes the following clauses:

Clause 6.4.1 - Stakeholder Requirements Definition Process
Clause 6.4.2 - Requirements Analysis Process
Clause 6.4.3 - Architectural Design Process
Clause 6.4.4 - Implementation Process
Clause 6.4.5 - Integration Process
Clause 6.4.6 - Verification Process
Clause 6.4.7 - Transition Process
Clause 6.4.8 - Validation Process
Clause 6.4.9 - Operation Process
Clause 6.4.10 - Maintenance Process
Clause 6.4.11 - Disposal Process

Design Process



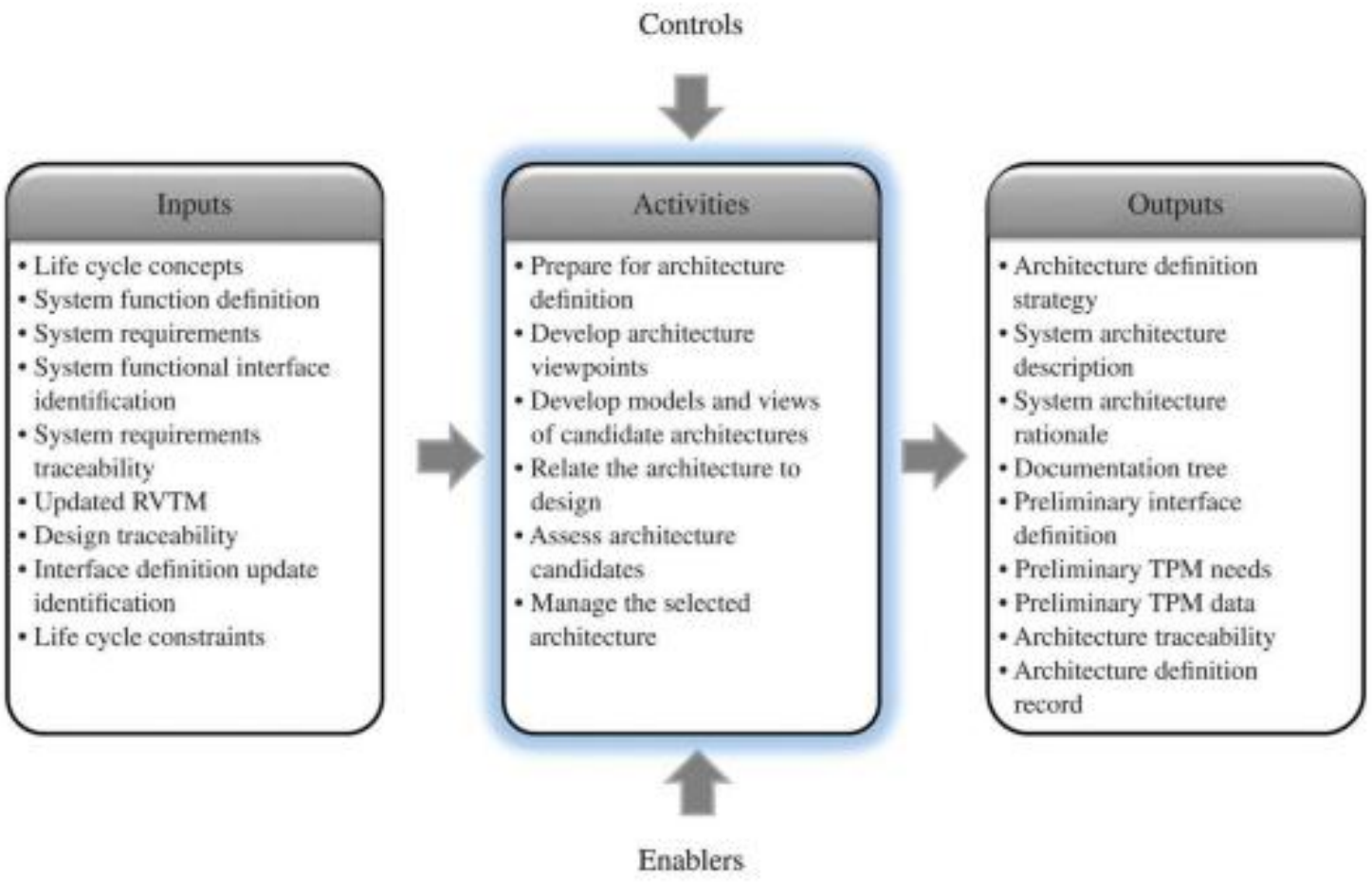
-NASA Systems Engineering Handbook

FIGURE 4.0-1 Interrelationships among the System Design Processes

ARCHITECTURE DEFINITION

As stated in ISO/ IEC/ IEEE 15288, [6.4.4.1] The purpose of the Architecture Definition process is to generate system architecture alternatives, to select one or more alternative(s) that frame stakeholder concerns and meet system requirements, and to express this in a set of consistent views.

ARCHITECTURE DEFINITION

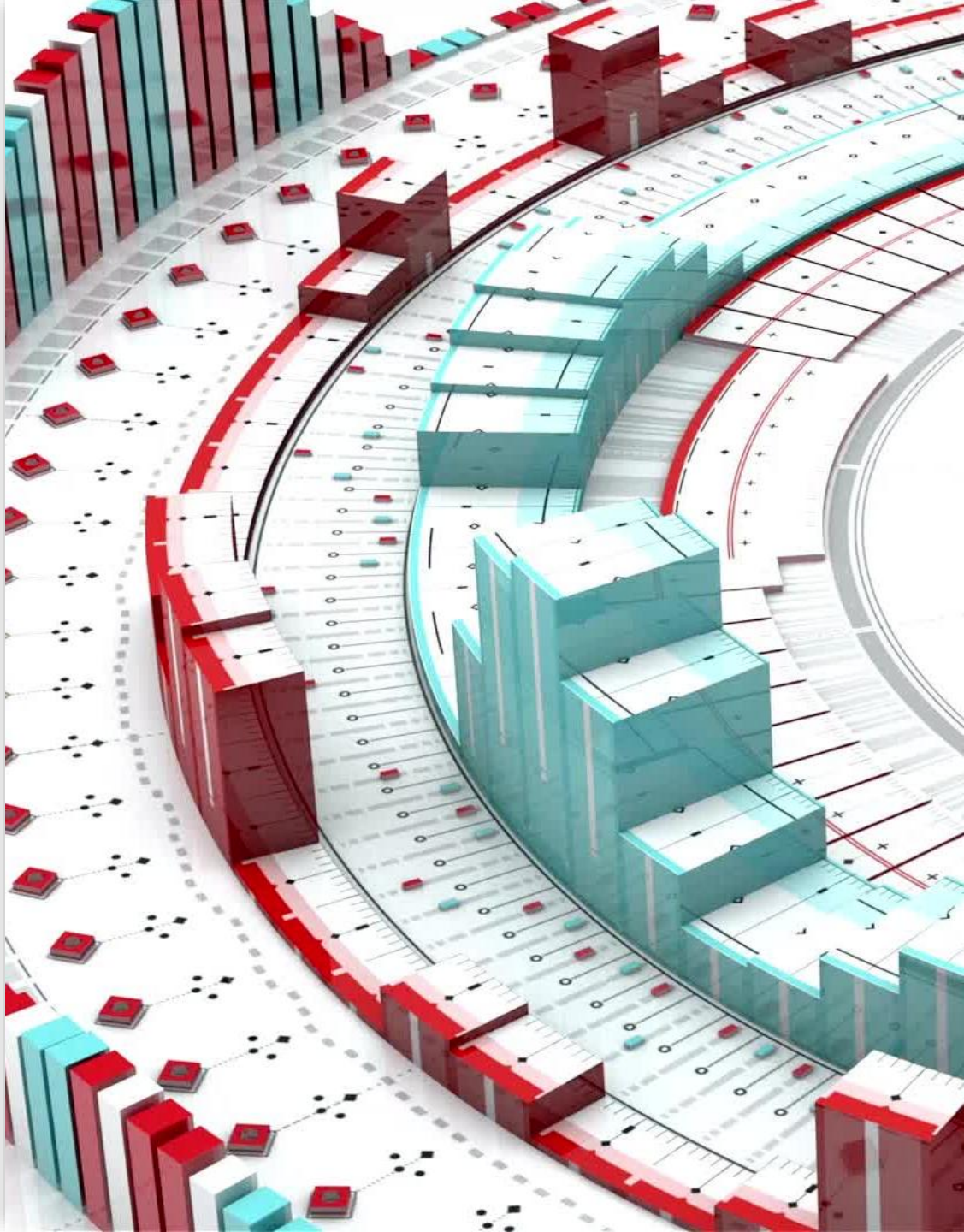


DESIGN DEFINITION PROCESS

- As stated in ISO/ IEC/ IEEE 15288, [6.4.5.1] The purpose of the Design Definition process is to provide sufficient detailed data and information about the system and its elements to enable the implementation consistent with architectural entities as defined in models and views of the system architecture.

PHYSICAL ARCHITECTURE

- A physical architecture model is an arrangement of physical elements, (system elements and physical interfaces) that provides the solution for a product, service, or enterprise. It is intended to satisfy logical architecture elements and system requirements ISO/IEC/IEEE 26702 (ISO 2007). It is implementable through technological system elements. System requirements are allocated to both the logical and physical architectures. The resulting system architecture is assessed with system analysis and when completed becomes the basis for system realization.
- -https://sebokwiki.org/wiki/Physical_Architecture



A DESIGN PROPERTY

- A design property is a property that is obtained during system architecture and created through the assignment of non-functional requirements, estimates, analyses, calculations, simulations of a specific aspect, or through the definition of an existing element associated with a system element, a physical interface, and/or a physical architecture. If the defined element complies with a requirement, the design property will relate to (or may equal) the requirement. Otherwise, one has to identify any discrepancy that could modify the requirement or design property and detect any deviations. .- https://sebokwiki.org/wiki/Physical_Architecture

ACTIVITIES IN THE ARCHITECTURE PROCESS

1

Partition and allocate functional elements to system elements

2

Constitute candidate physical architecture models.

3

Assess physical architecture model candidates and select the most suitable one

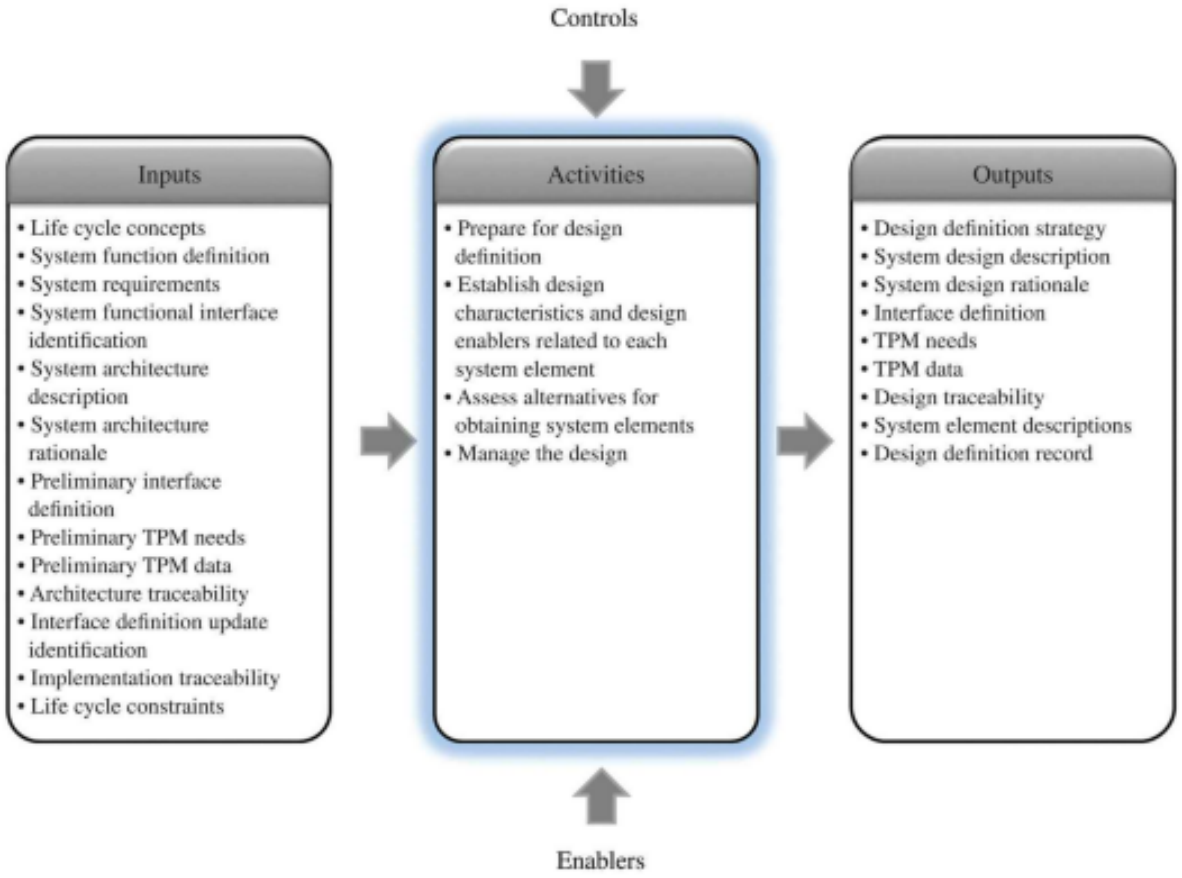
4

Synthesize the selected physical architecture model

ARTIFACTS OF THE ARCHITECTURE PROCESS

- Physical block diagrams (PBD)
- SysML block definition diagrams (BDD)
- Internal block diagrams (IBD) (OMG 2010)
- Executable architecture prototyping

DESIGN DEFINITION PROCESS



CRITICAL DESIGN REVIEW

A multi-disciplined technical review, conducted at both system-level and component-level, ensures that the initial product baseline is established. The component-level CDRs should be successfully completed for each major component before conducting the system-level CDR. It completes the process of defining the technical requirements for each component, which are documented in the item performance specification of each component. A successful completion of CDR provides a sound technical basis for proceeding into fabrication, integration, and developmental test and evaluation. At completion of the CDR, the *initial product baseline* is normally taken under contractor configuration control at least until the physical configuration audit (PCA).

ANALYSIS OF ALTERNATIVES (AOA)

The AoA assesses potential materiel solutions to mitigate the capability gaps documented in the validated Initial Capabilities Document (ICD). The AoA focuses on identification and analysis of alternatives, measures of effectiveness (MOE), cost, schedule, concepts of operation, and overall risk. This includes the sensitivity of each alternative to possible changes in key assumptions or variables. The AoA addresses trade space to minimize risk and also assesses critical technology elements associated with each proposed materiel solution. This includes technology maturity, integration risk, manufacturing feasibility, and, where necessary, technology maturation and demonstration needs. The AoA normally occurs during the Materiel Solution Analysis (MSA) phase of the Acquisition process, is a key input to the Capability Development Document (CDD), and supports the materiel solution decision at Milestone A. (Sources: DoDI 5000.02 and JCIDS Manual)

ANALYSIS OF ALTERNATIVES (AOA)

AoA Study Guidance - For potential and designated ACAT I programs, the Director for Cost Assessment and Program Evaluation (DCAPE) prepares study guidance for the DoD Component Head 40 business days prior to the Materiel Development Decision (MDD). The Milestone Decision Authority (MDA) will include the study guidance with the MDD acquisition decision memorandum. For ACAT II and III programs, Component AoA procedures apply.

AoA Study Plan - For ACAT I programs, the AoA Study Plan is developed by the component or agency from the AoA Study Guidance issued by DCAPE and submitted 20 days prior to the MDD. For ACAT I programs, the AoA study plan must be approved by DCAPE. The AoA Plan details the approach the sponsor will follow when conducting the AoA during the Mission Solutions Analysis (MSA) phase. In addition to building the plan and submitting it for approval, the component or agency must submit a memorandum to the MDA for ACAT I programs that ensure the completion of the AoA within nine months. Only the Secretary of Defense or his/her delegate can waive this nine-month window.

ANALYSIS OF ALTERNATIVES (AOA)

AoA Final Report – The final Study Advisory Group (SAG) will meet either 55 business days before the next milestone's request for proposal (RFP) or no later than nine months after the start of the AoA. The AoA Sponsor provides the final AoA to DCAPE no later than 40 business days after briefing the final SAG. DCAPE evaluates the AoA and provides a memorandum to the MDA no later than 40 business days after receiving the AoA Final Report from the component (this memorandum may be received after the Developmental Request for Proposal Release Decision (DRFPRD) or Milestone A. The sponsor also sends copies to the DoD Component head or other organization or principal staff assistant assessing whether the analysis was completed consistent with DCAPE Study Guidance and the DCAPE-approved Study Plan)

ALTERNATIVE SYSTEM REVIEW (ASR)

The Alternative Systems Review (ASR) should be conducted typically after the completion of the analysis of alternatives (AoA). The system parameters should be defined based on the trade off among cost, schedule, and risks.

The inputs of an ASR include:
- The AoA results.
- The preferred materiel solution.
- The draft concept of operations (CONOPS).

The outputs of an ASR include:
- Refined description of the preferred materiel solution to support further development.
- Informed advice to the user-developed draft Capability Development Document (CDD) required at Milestone A.

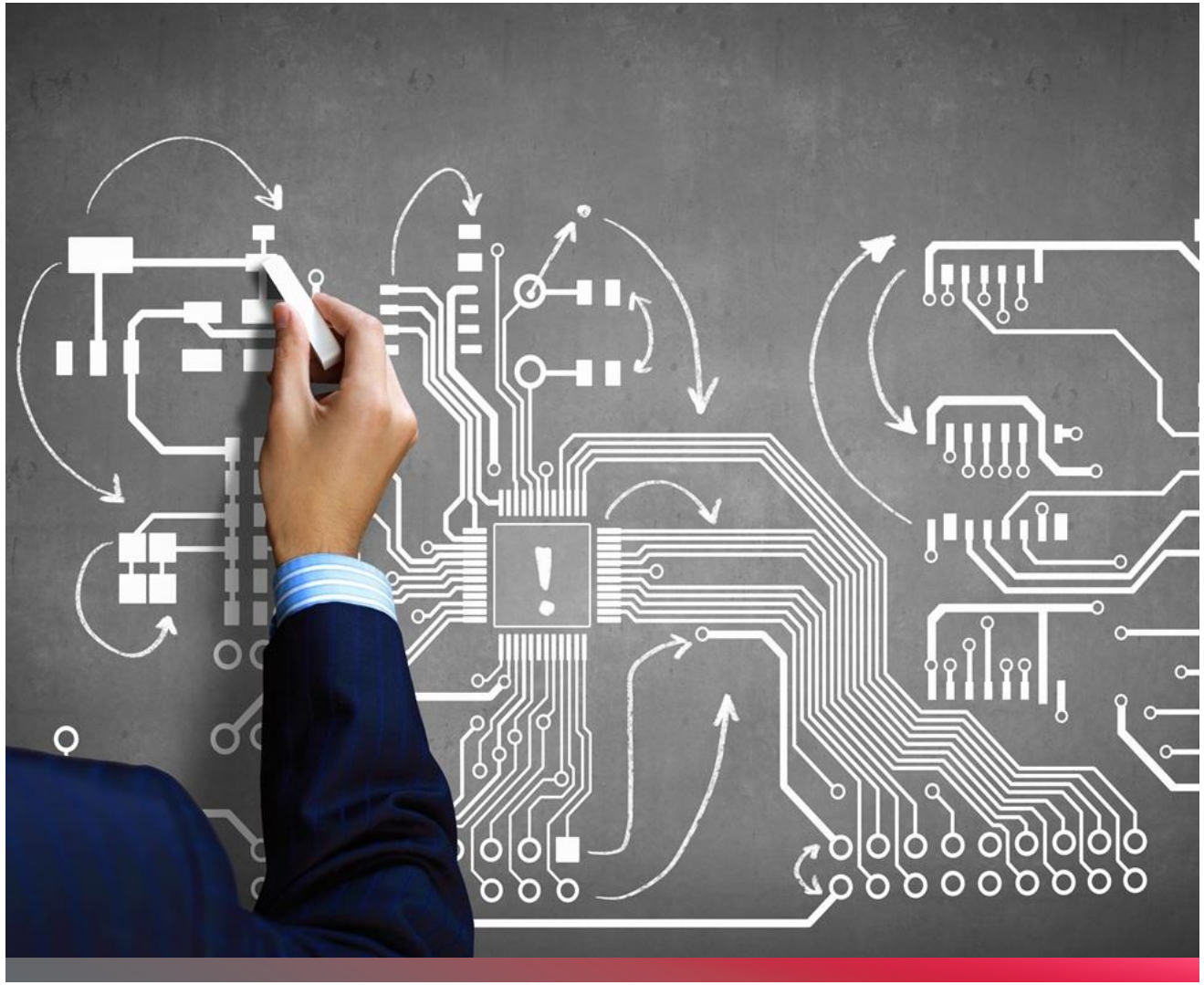
CAPABILITIES BASED ASSESSMENT (CBA)

A Joint Capabilities Integration and Development System (JCIDS) analytic process. The CBA identifies capability requirements and associated capability gaps. Results of a CBA or other study provide the source material for one or more Initial Capabilities Documents (ICDs), or other JCIDS documents in certain cases when an ICD is not required

A number of DoDAF views are to be used to capture results of a CBA, facilitating reuse in JCIDS documents, acquisition activities, and capability portfolio management. When one or more studies or analyses are used in place of a CBA, the Sponsor may need to consolidate the data from those studies into a single set of DoDAF products appropriate for the scope of the ICD.

Design for Diagnosability

- Design for Diagnosability is a systems engineering and reliability principle that focuses on making a system easy to detect, isolate, and identify faults during operation and maintenance. It’s a proactive design approach—integrated from the early stages—to ensure that when something goes wrong, you can quickly and accurately figure out what and where the problem is.



Design for Diagnosability

Design Techniques

- **Built-In Test Equipment (BITE)**

Integrated electronics or software that perform self-tests and report failures.

- **Modularization**

Design the system so faulty modules can be quickly replaced (Line Replaceable Units, LRUs).

- **Clear Feedback & Alerts**

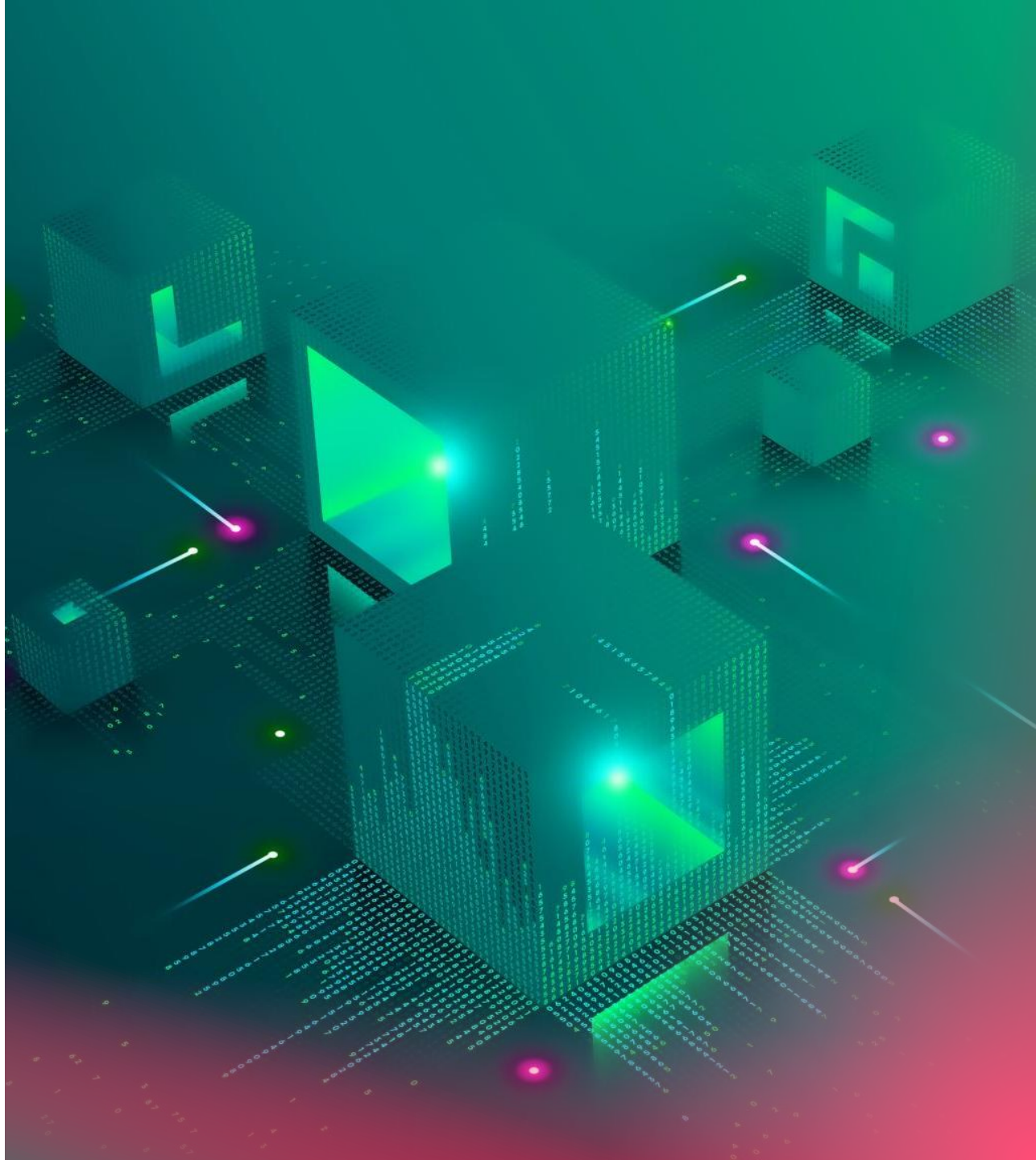
Use standardized fault codes, graphical interfaces, or verbal alerts to convey what failed and why.

- **Redundancy with Isolation**

Not just having backups, but also the ability to tell which path/component failed.

- **Accessible Test Points**

Place connectors, pads, or software hooks to measure critical signals without dismantling the system.



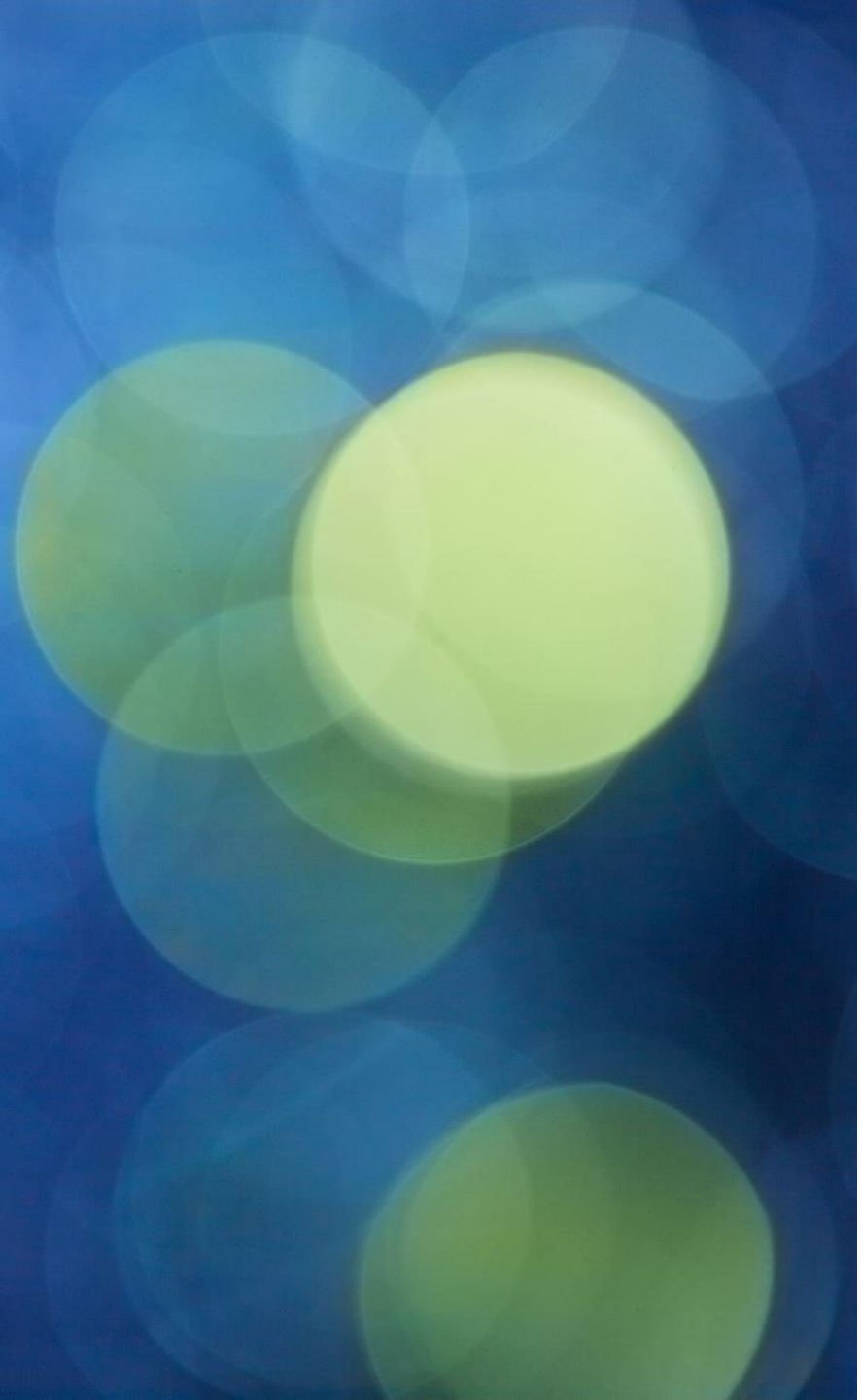
Design for Diagnosability

- Consider a UAV system as an example. A UAV propulsion system is designed for diagnosability by:
  - Embedding ESC telemetry (RPM, current, temperature) into the flight controller logs.
  - Using vibration sensors near motors to detect early bearing wear.
  - Assigning unique error codes for motor overheat vs. ESC overcurrent.
  - Designing the wiring harness with color-coded quick-disconnects for fast replacement

PART II
VERIFICATION AND VALIDATION

Verification vs validation

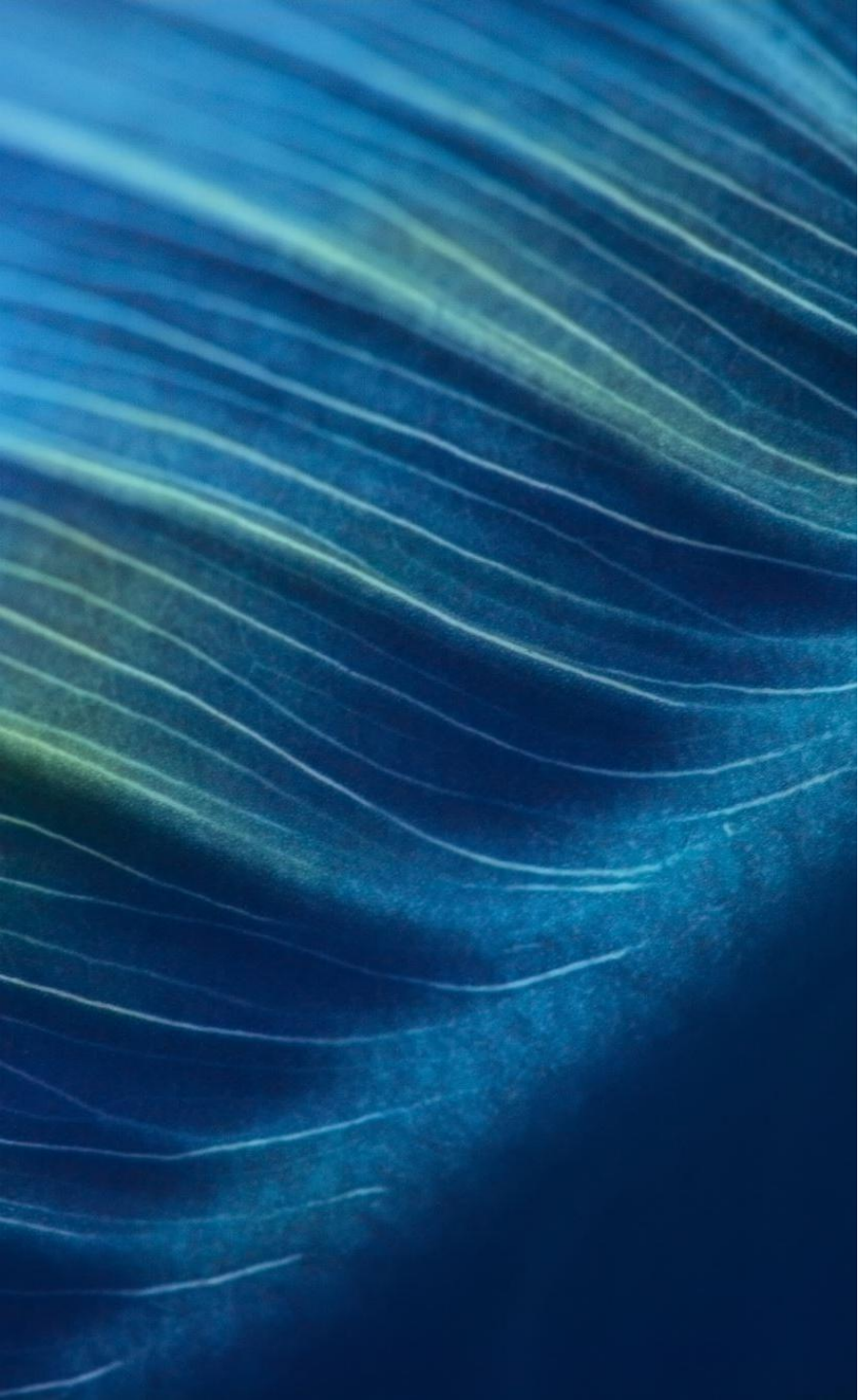
- Verification:
  "Are we building the product right”.
- The software should conform to its specification.
- Validation:
  "Are we building the right product”.
- The software should do what the user really requires.



Definitions

V&V – a system engineering discipline employing a rigorous methodology for evaluating and assessing the correctness and quality of software throughout the software life cycle.

Verify a developers process is technically sound.



V&V and QA

V&V and QA are not the same, but compliment each other.

V&V usually focuses on ensuring the requirements are being met, the overall project is focused on the correct objectives, and risk is being managed.

QA is focused on the day to day aspects of a project and is used to determine if procedures are followed

V&V Concepts

- Early detection leads to a better solution rather than quick fixes
- Validating the solution is solving the “right problem” against software requirements
- Objective evidence of software and system compliance to quality standards
- Support process improvements with an objective feedback on the quality of development process and products

V & V goals

- Verification and validation should establish confidence that the software is fit for purpose
- This does NOT mean completely free of defects
- Rather, it must be good enough for its intended use and the type of use will determine the degree of confidence that is needed

V & V planning

Careful planning is required to get the most out of testing and inspection processes

Planning should start early in the development process

The plan should identify the balance between static verification and testing

Test planning is about defining standards for the testing process rather than describing product tests

Planning for V&V

Scope of work

Software Integrity Levels

Development of the Software V&V Plan (SVVP)

Cost of V&V

Planning for V&V (cont)

V&V is more effective when initiated during the acquisition process and throughout the life cycle of the software.

V&V has importance levels or called “Integrity Levels”

Example

- Medical device – high level
- Personnel record-keeping system – low level

Integrity Levels

The level is a range of values that represent software complexity, criticality, risk, safety level, security level, desired performance, reliability, or other project-unique characteristics.

Each level defines the minimum required V&V tasks.

ANSI/IEEE Std 1012 defines four levels. Level 4 is assigned to high-assurance or critical systems

Life Cycle V&V Tasks

Acquisition V&V

Supply V&V

Development V&V (Concept, Requirements, Design, Implementation)

Development V&V (Test)

Development V&V (Installation and Checkout)

Operation V&V

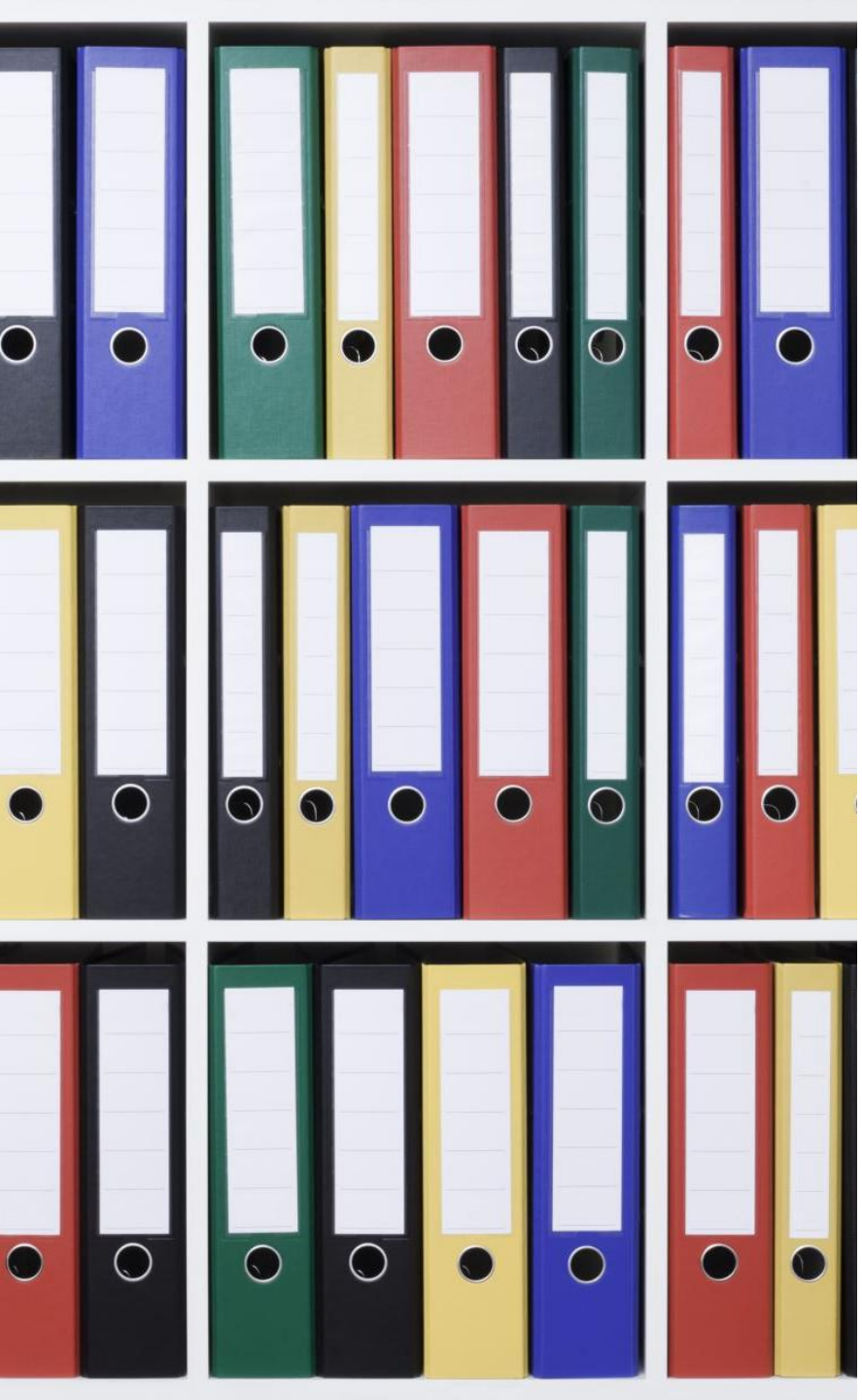
Maintenance V&V

V&V Techniques and Methods

Audits, Reviews, and Inspection

Analytic Techniques

Dynamic Techniques



Audits, Reviews, and Inspection

- V&V use these techniques to verify the software during its development process
  - Peer Reviews
  - Documentation inspections
  - Requirements/design/code reading
  - Test witnessing
  - Installation audits

Analytic Techniques

Static analysis of the software (i.e, requirements, design, or code) using graphical, mathematical formulas or diagrams.

Effective in error detection at the software unit level

Analytic Techniques

Control (data) flow diagramming

Interface input/output/process diagramming

Algorithm and equation analysis

Database analysis

Sizing and timing analysis

Proof of correctness

What to look for

- The number of iterations the algorithm requires. In any algorithm, the process will need to be repeated (at least for sorting or searching algorithms). The number of iterations required, both on average, and worst case, is one way to measure the efficiency of the algorithm.

- The amount of resources required. Any algorithm will require a certain amount of resources, usually RAM (Random Access Memory). The resources required, is yet another way to evaluate the efficiency of an algorithm.

Big O Analysis

- Perhaps the most common way to formally evaluate the efficacy of a given algorithm is Big O notation (Mathworld 2005). This method is measure of the execution of an algorithm, usually the number of iterations required, given the problem size n. In sorting algorithms n is the number of items to be sorted. . Stating some algorithm f(n) = O(g(n)) means it is less than some constant multiple of g(n). The notation is read, "f of n is big oh of g of n". This means that saying an algorithm is 2N, means it will have to execute 2 times the number of items on the list. Big O notation essentially measures the asymptotic upper bound of a function. Big O is also the most often used analysis.
- Big O Notation was first introduced by the mathematician Paul Bachmann in his 1892 book *Analytische Zahlentheorie*. The notation was popularized in the work of another mathematician named Edmund Landau. Because Landaue was responsible for popularizing this notation. it is sometimes referred to as a Landau symbol.

Big O Analysis

- Omega notation is the opposite of Big O notation. It is the asymptotic lower bound of an algorithm and gives the best-case scenario for that algorithm. It gives you the minimum running time for an algorithm (Cormen 2001).

- Theta notation combines Big O and Omega to give the average case (average being arithmetic mean in this situation) for the algorithm. In our analysis we will focus heavily on the Theta, also often referred to as the Big O running time. This average time gives a more realistic picture of how an algorithm executes (Cormen 2001).

- Note: It can be confusing when a source refers to a Big O running time other than a theta. In writing this paper I found several online sources that used O notation, when clearly what they where providing was actually .

Space Complexity

This topic is defined in different ways in different sources. One common definition is the amount of memory space required to solve an instance of the problem, as a function of the input. Time complexity is simply the amount of time it takes to execute an algorithm. Obviously, these both relate to what is often called time-space complexity.

Space complexity is normally considered to have two different spaces: Input space and Auxiliary space. As the name suggests, auxiliary space is that space needed, other than what is needed for input.

FYI, my recommendation for a basic algorithm book has always been *Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L.; Stein, Clifford. Introduction to Algorithms, third edition. MIT Press.*

For something more advanced: *Peter Bürgisser. Completeness and Reduction in Algebraic Complexity Theory (Algorithms and Computation in Mathematics)*

Cyclomatic Complexity

In 1976 Thomas McCabe developed cyclomatic complexity (McCabe, 1976). Cyclomatic complexity is defined as the number of linearly independent paths in a given body of code (Pawade, Dave, & Kamath, 2016). Thus, if there are no code branches such as in if statements, switch statements, or other decision points, then there is a cyclomatic complexity of 1. There exists only one linearly independent path through the code (Ukić, Maras, & Šerić, 2018). Put more formally, the cyclomatic complexity of source code is defined using a control flow graph of the program or function (Ebert & Cain, 2016). This is a directed graph wherein the nodes are the basic blocks of the program or function and the edges connect those nodes. The cyclomatic complexity is defined as:

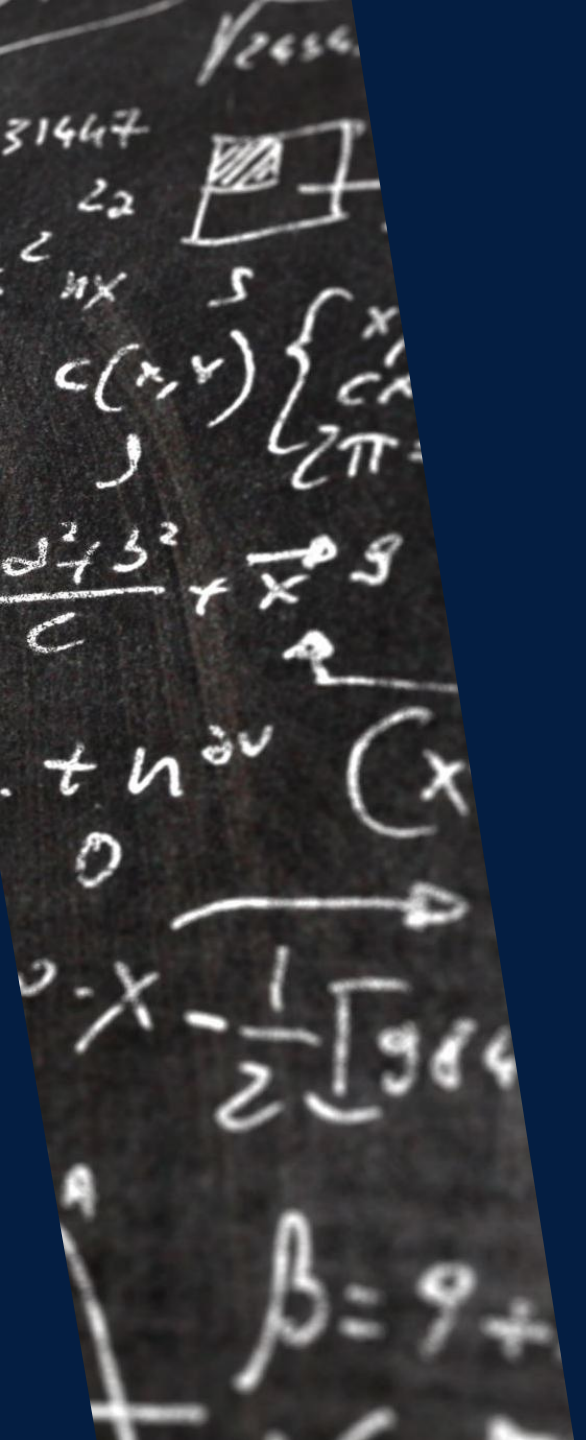
$C = E - N + 2P$, where

E = the number of edges of the graph.

N = the number of nodes of the graph.

P = the number of connected components

This is a useful approach for measuring complexity because it integrates graph theory. McCabe only integrated the most basic elements of graph theory (McCabe, 1976), however, once one has expressed a problem as a graph, the full power of graph theory could be applied. Put another way, it would not be an overly arduous task to expand this definition to incorporate a range of graph theory elements such as weighted nodes and edges, incidence functions relating the nodes, or even more sophisticated aspects of graph theory such as spectral graph theory (Easttom, 2020b).



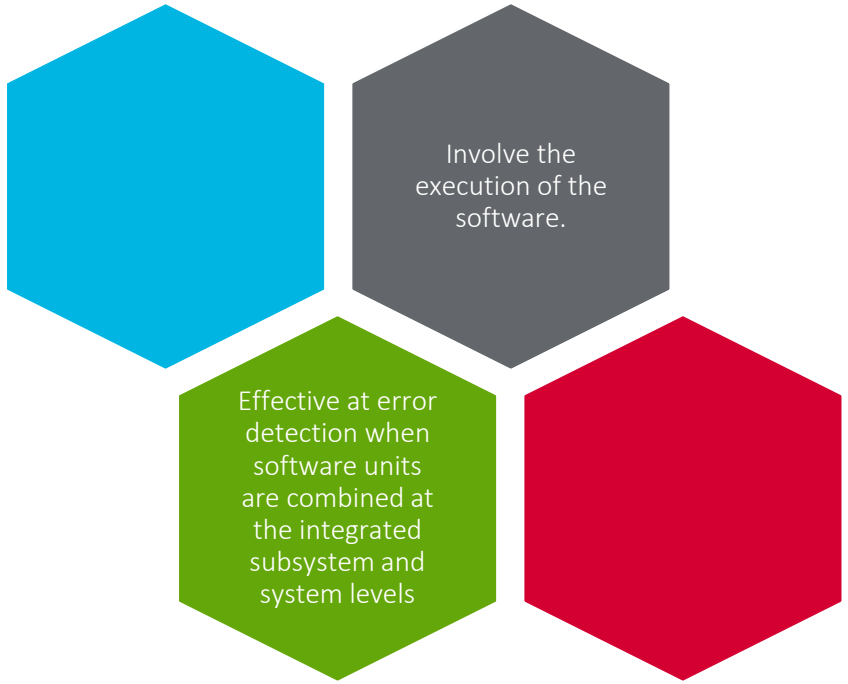
Halstead Metrics

The field of computational complexity is rather robust and well developed. This is evident from the multiple modalities of studying computational complexity. Within that context there are several methods for calculating complexity. In 1977 Maurice Halstead put forth what are now known as Halstead complexity measures or metrics .These are metrics for measuring software complexity. Halstead posited several metrics for software complexity (shown in table 3.

Halstead Metrics

Metric	Value
n1	Number of distinct operators.
n2	Number of distinct operands.
N1	Total number of occurrences of operators.
N2	Total number of occurrences of operands.
n1*	Number of potential operators.
n2*	Number of potential operands.

Dynamic Techniques



Dynamic Techniques

- Simulation and modeling
- Hardware/software benchmark testing
- Hardware-in-the loop testing – the system config. is heavily instrumented to simulate different test scenarios to be created.
- Scientific testing – coding of the target requirements/design using a general-purpose computer and higher order language.

Measurement Applied to V&V

- Uses various calculated measurements to determine when the analysis or testing is completed, where errors are mostly likely to occur in the software, and what development process or function is causing the largest number of errors.
- Based on these measurements, the software engineer can determine where to concentrate their efforts.

Measurement Methods

- Software Structural Metrics – measures pinpoint program logic having greater logical or data complexity
- Statistics-Based Measurements – examines program error rates, categorization of errors, and error discovery time periods
- Trend Analysis – analyzing percent of errors with historical data
- Prediction-Based Measurement – using reliability models to determine how much analysis and test effort to be done.

Acronyms

Terms	Meaning
MTBF	Mean Time Between Failures
MTTR	Mean Time To Repair (corrective maintenance only)
MTBMA	Mean Time Between Maintenance Actions (corrective and preventive maintenance)
MMT	Mean Maintenance Time (corrective and preventative maintenance)
MDT	Mean Downtime (includes downtime due to active maintenance and logistics delays)

Engel, Avner. Verification, Validation, and Testing of Engineered Systems (Wiley Series in Systems Engineering and Management Book 73)

Availability

Inherent Availability: System availability assuming corrective maintenance is only undertaken when the system fails

IA = MTBF/ (MTBF+MTTR)

Achieved Availability: System availability assuming maintenance is undertaken for both corrective and preventive actions and all logistics

AA = MTBMA/(MTBMA +MNT)

Operational Availability: System availability assuming maintenance is undertaken for both corrective and preventive actions and average logistic delays are encountered:

OA = MTBMA/(MTBMA + MDT)

Engel, Avner. Verification, Validation, and Testing of Engineered Systems (Wiley Series in Systems Engineering and Management Book 73)

PART III

More Modeling

Views and Viewpoints

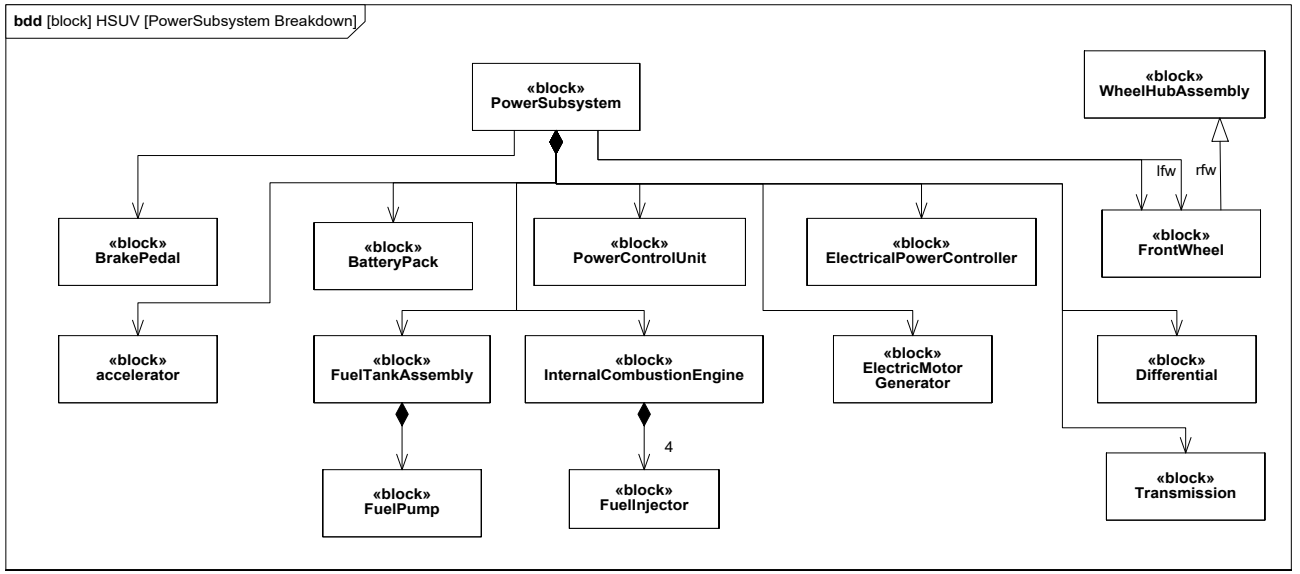
- Consistent with IEEE 1471
- Viewpoint represents stakeholders, their concerns/purpose/intent, construction rules for specifying a view
- View is a read only mechanism that captures the model subset that addresses the stakeholder concerns
  - Realizes the viewpoint
  - Relationships between model elements established in model and not between views

IEEE 1471

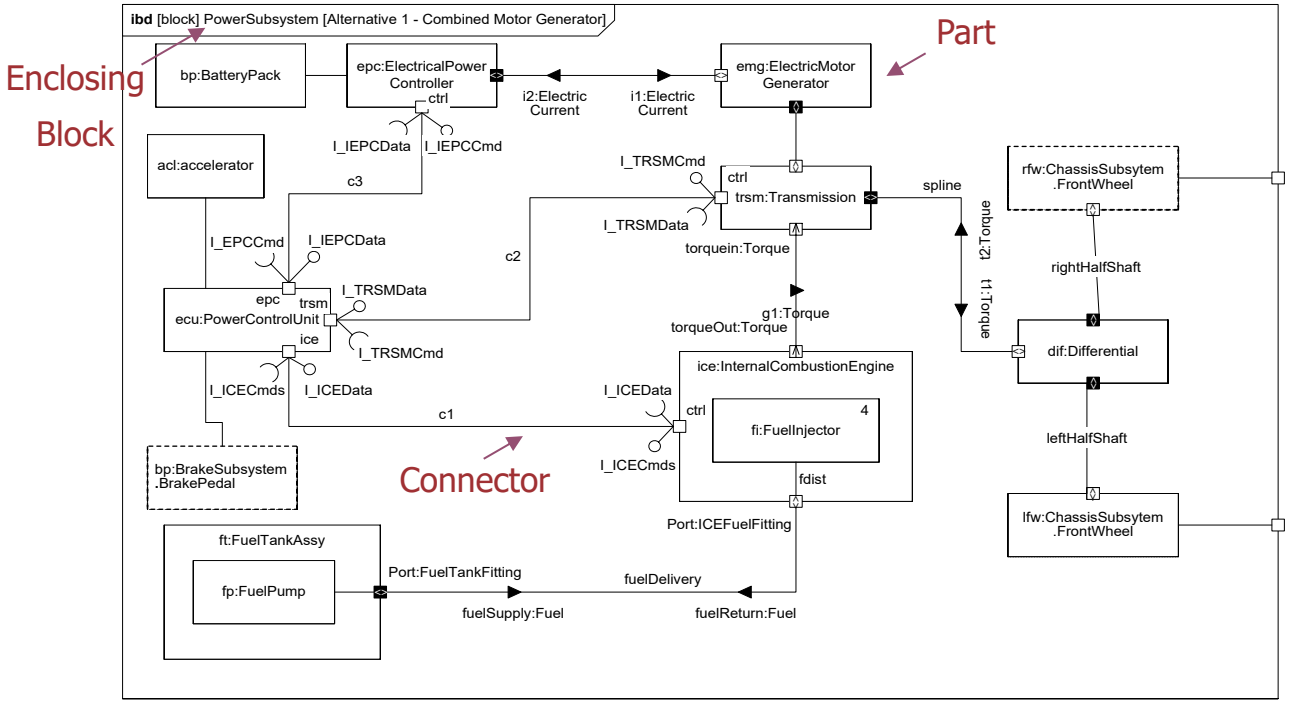
- IEEE 1471 (section 5.3) prescribes that a viewpoint contains:
  - a) A viewpoint name
  - b) The stakeholders to be addressed by the viewpoint
  - c) The concerns to be addressed by the viewpoint
  - d) The language, modeling techniques, or analytical methods to be used in constructing a view based upon the viewpoint
  - e) The source, for a library viewpoint (the source could include author, date, or reference to other documents, as determined by the using organization)

Power Subsystem Breakdown

- Block Definition Diagram Used to Specify System Hierarchy and Classification

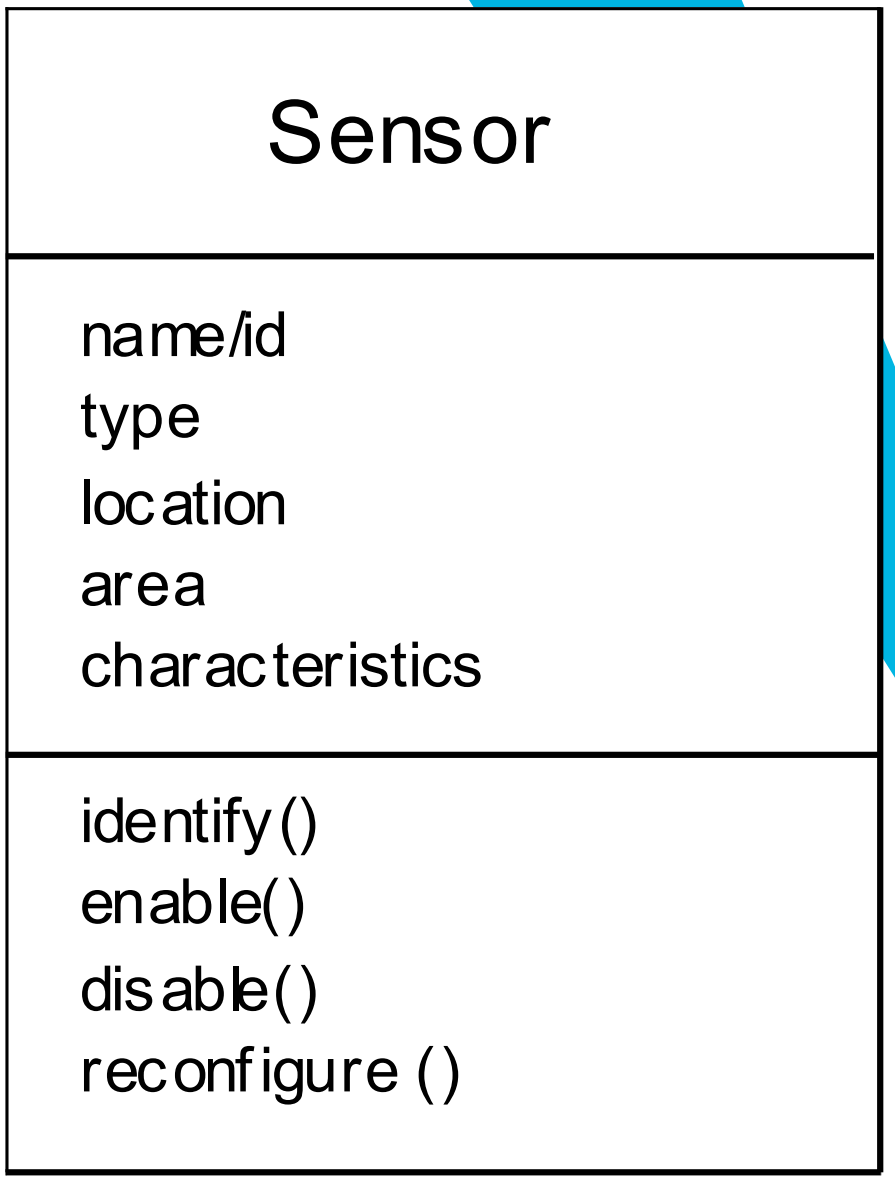


Power Subsystem IBD



Internal Block Diagram Used to Specify Interconnection Among Parts in Context of Enclosing Block

UML CLASS DIAGRAM



UML STATE DIAGRAM

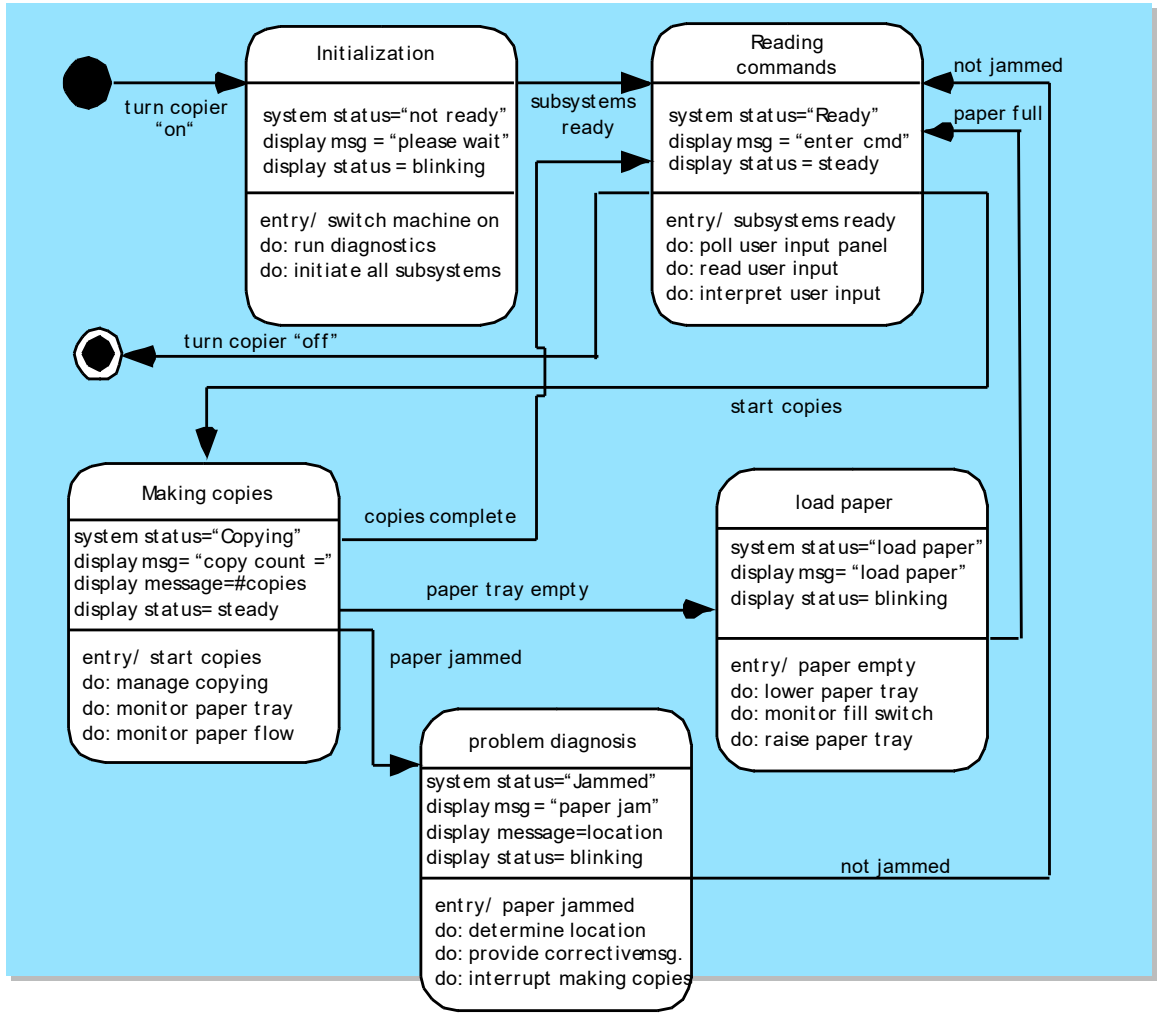
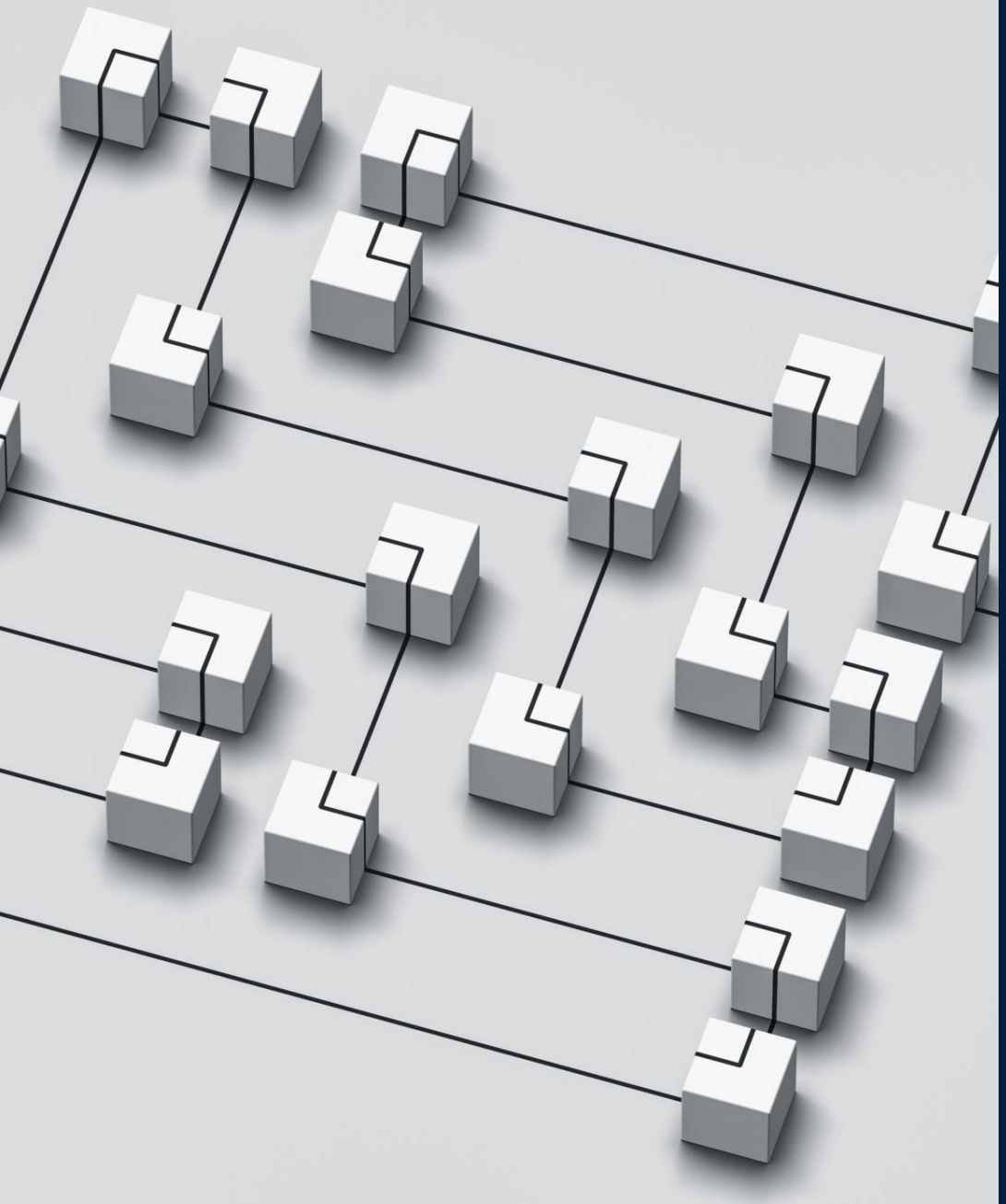


Figure 7.6 Preliminary UML state diagram for a photocopier



Ports Approach

- Ports represent block interaction points via which
  or consume data/material/energy or services
- Support specification of interfaces on a block indep
  specific usage (e.g. this component requires 110 vol
  input)
- Approach is to specialize two port types
  - Flow ports
    - Port type specifies what can flow in our out of
      block/part
    - A connection point through which there is a flow
      information, material, or energy (I/O)
    - Typically asynchronous flow where producer is not
      aware when/who consumes the flow
  - Client server ports
    - Service oriented (request-reply) peer2peer interactio
    - Typically synchronous communication
    - Specified similar to UML2.0 ports using
      required/provided interfaces detailing the set of
      provided/required services
    - Allow signal exchanges for compatibility

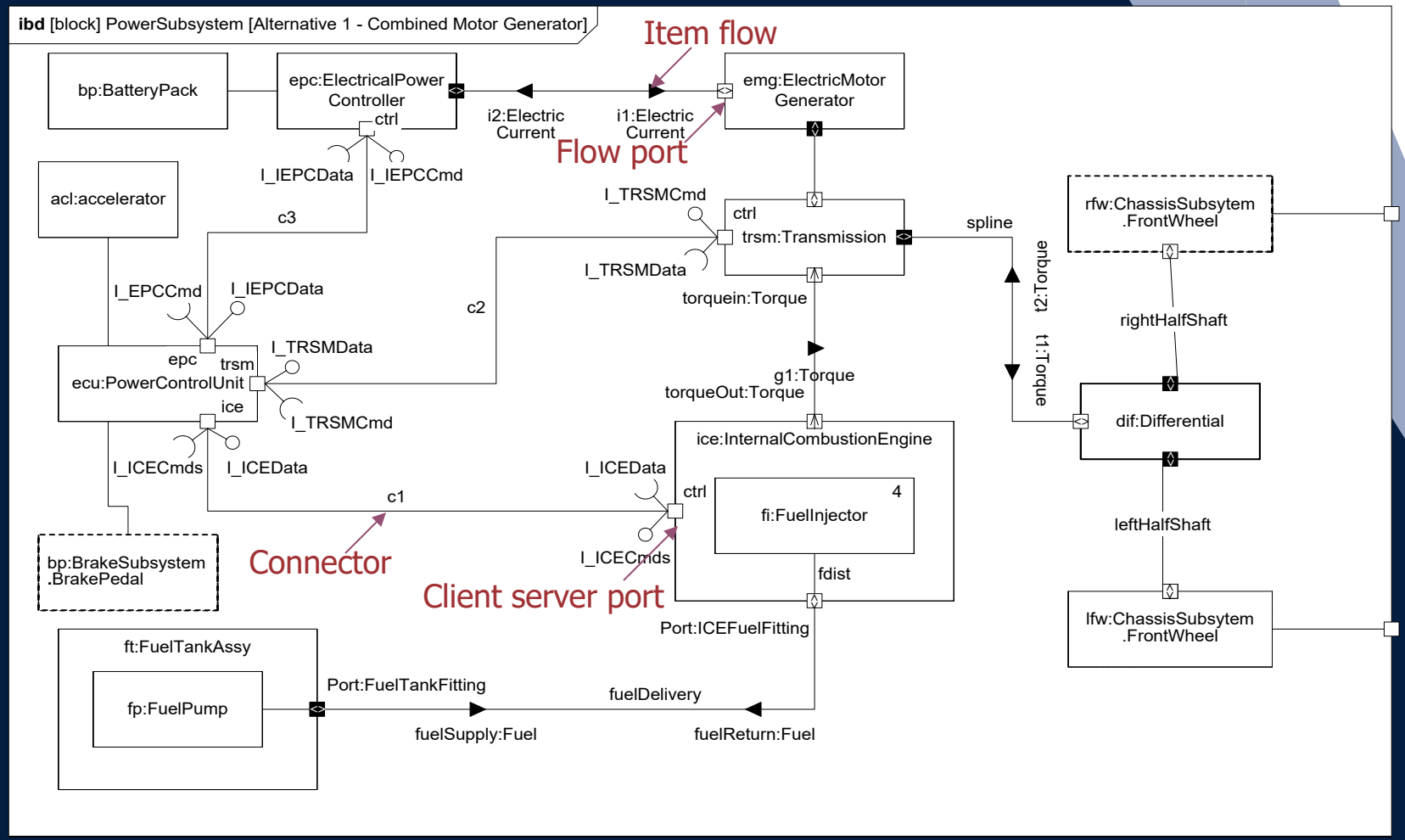
FlowPorts

- Additional considerations
  - Simple (natural) way for SEs to specify I/O via the port
  - Address the common case of atomic FlowPorts
  - Allow both signal flow and data/block instance flow
- FlowPorts Specification
  - I/O is specified using an interface stereotyped FlowSpecification
  - FlowSpecification consists of properties stereotyped FlowProperties
    - FlowProperty has a direction attribute: in, out, inOut
    - FlowProperties can be typed by ValueTypes, Block, and Signals
    - isConjugate promotes reuse of flowSpecifications
- Atomic FlowPorts
  - It is common that a FlowPort flows a single item type
  - In this case the port is directly typed by the item type (Block or Value)
  - Direction property specify the direction
- Compatibility rules on ports facilitate interface compatibility

Item Flows Approac

- Distinct from what can flow via the port specification
- Supports compact and intuitive modeling physical flows
- Supports top down description of flows with imposing behavioral method (e.g. activities, state, interactions)
  - Is aligned with behavior thru refinement an allocation
- Facilitates flow allocations from an object node, message, or signal from a behavioral diagram
- Properties of item flow can be specified and constrained in parametric diagram

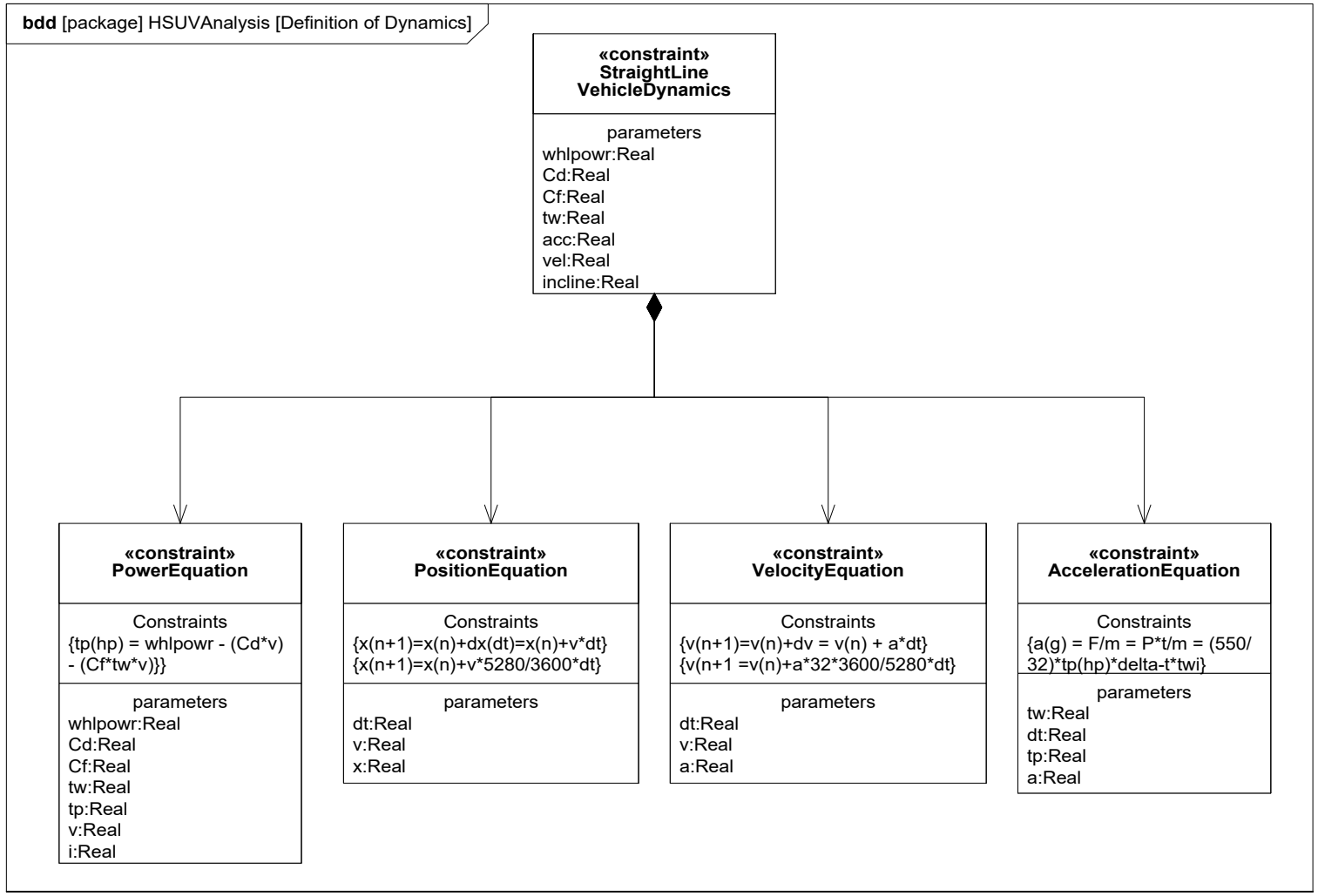
Power Subsystem IBD



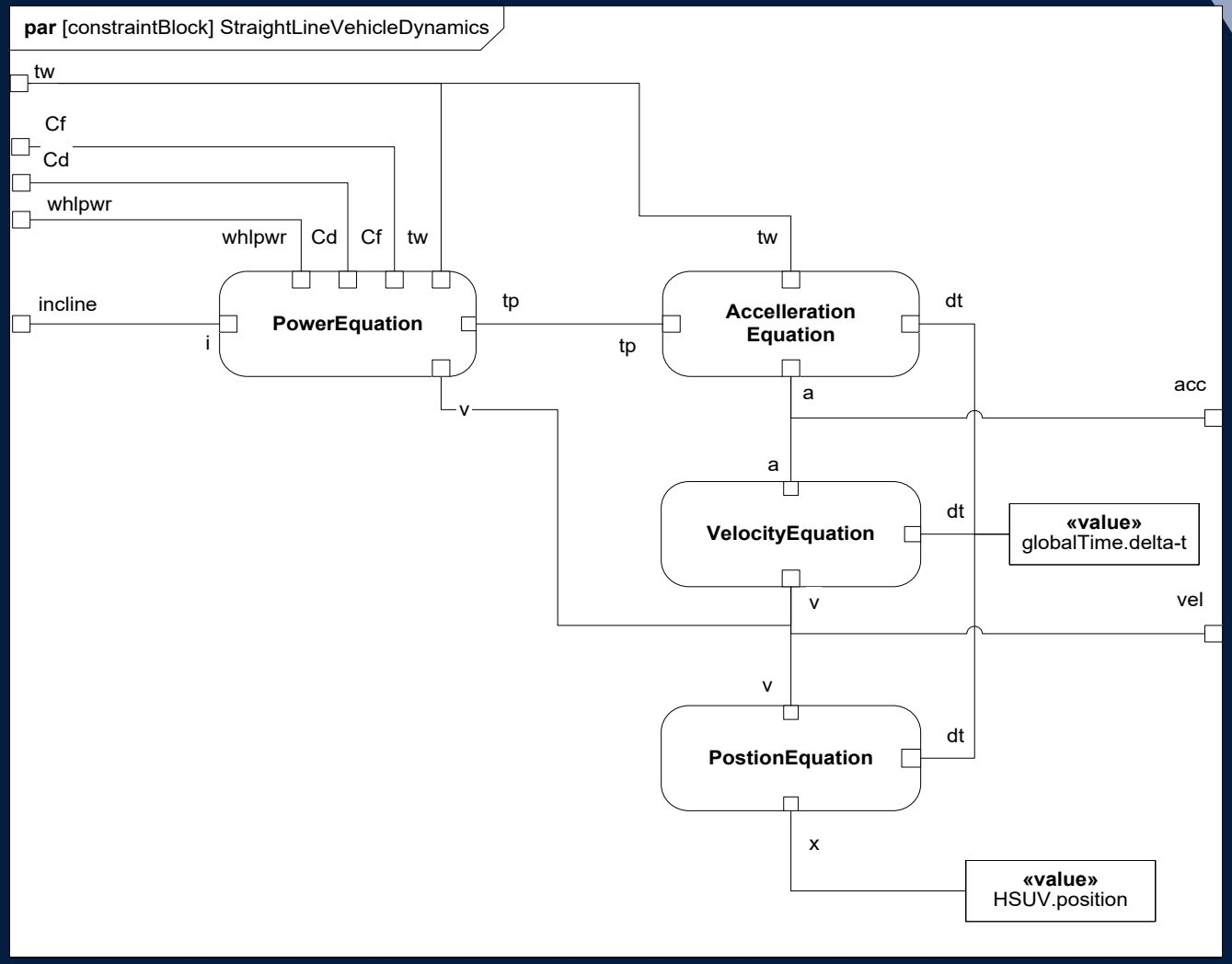
Parametrics

- Used to express constraints (equations) between value properties
  - Provides support to engineering analysis (e.g. performance, reliability, etc)
  - Reusable (e.g. F=m*a is reused in many contexts)
  - Non-causal (i.e. declarative statement of the invariant without specifying dependent/independent variables)
- Constraint block defined as a simple extension of block
  - Packages UML constraint so they are reusable and parameterized
  - Constraint and constraint parameters are specified
  - Expression language can be formal (e.g. MathML, OCL …) or informal
  - Computational engine is defined by applicable analysis tool and not by SysML
- Parametric diagram represents the usage of the constraints in an analysis context
  - Binding of constraint usage to value properties of blocks (e.g. vehicle mass bound to F= m * a)
    - Can use nested notation or dot notation
- MOE's and objective functions integrated with Parametrics to support trade studies and engineering analysis

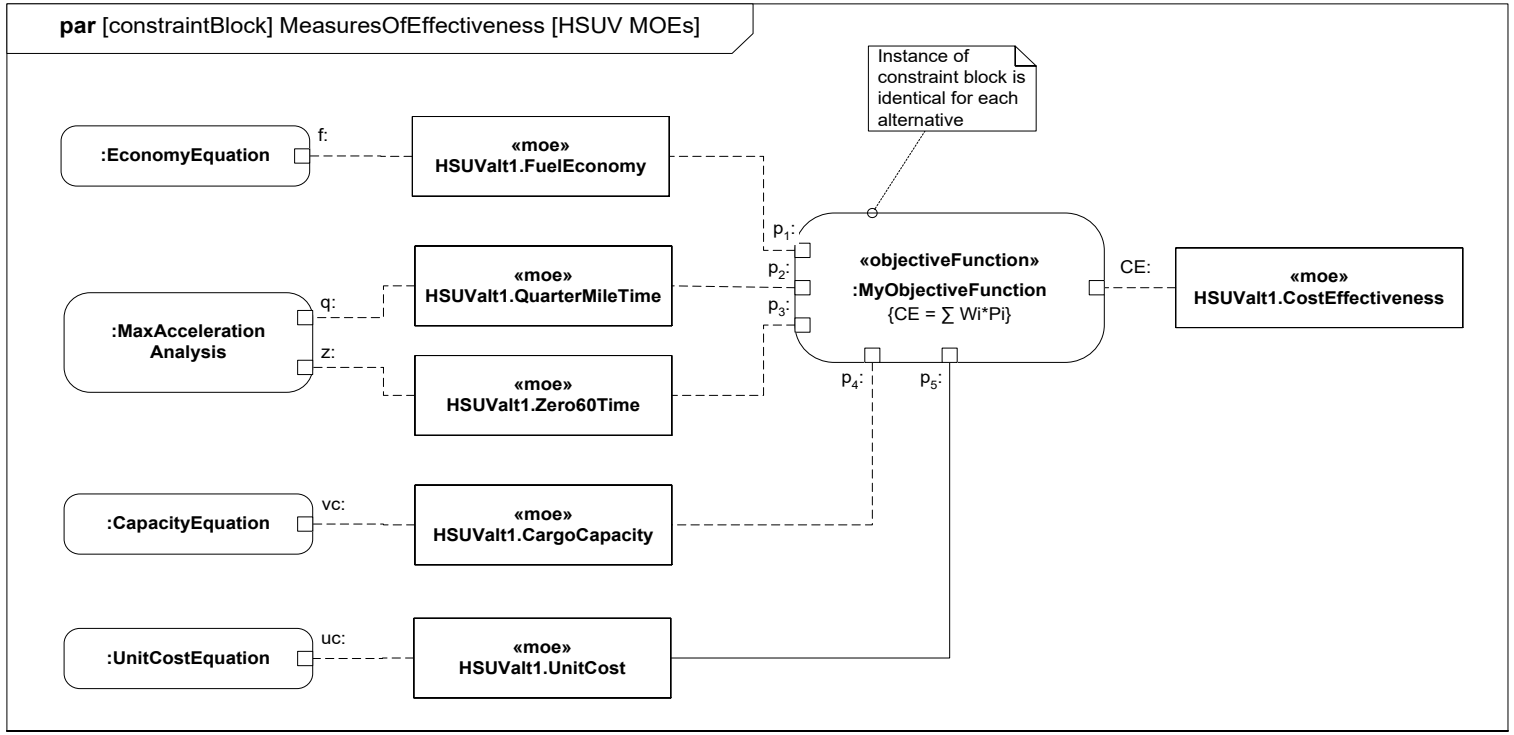
Defining Vehicle Dynamics



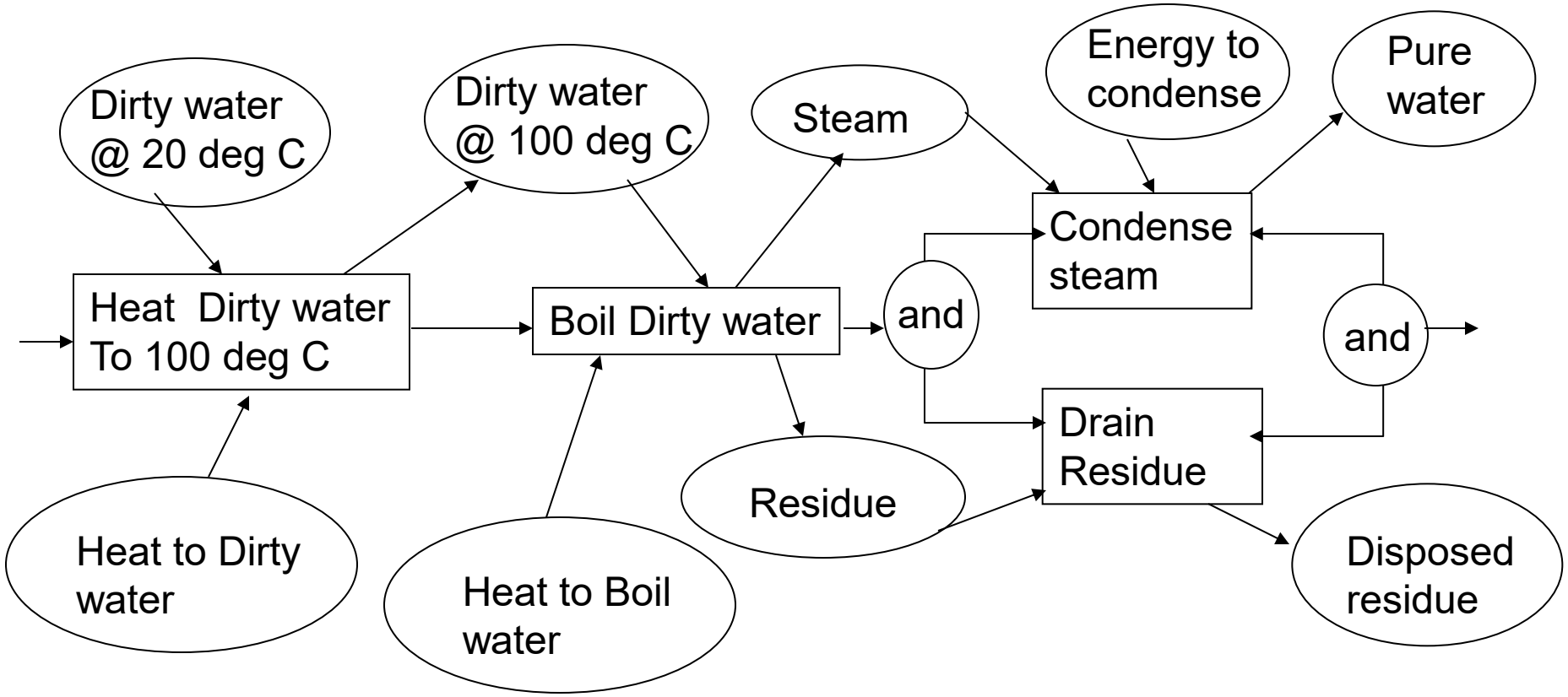
Evaluating Vehicle Dynamics



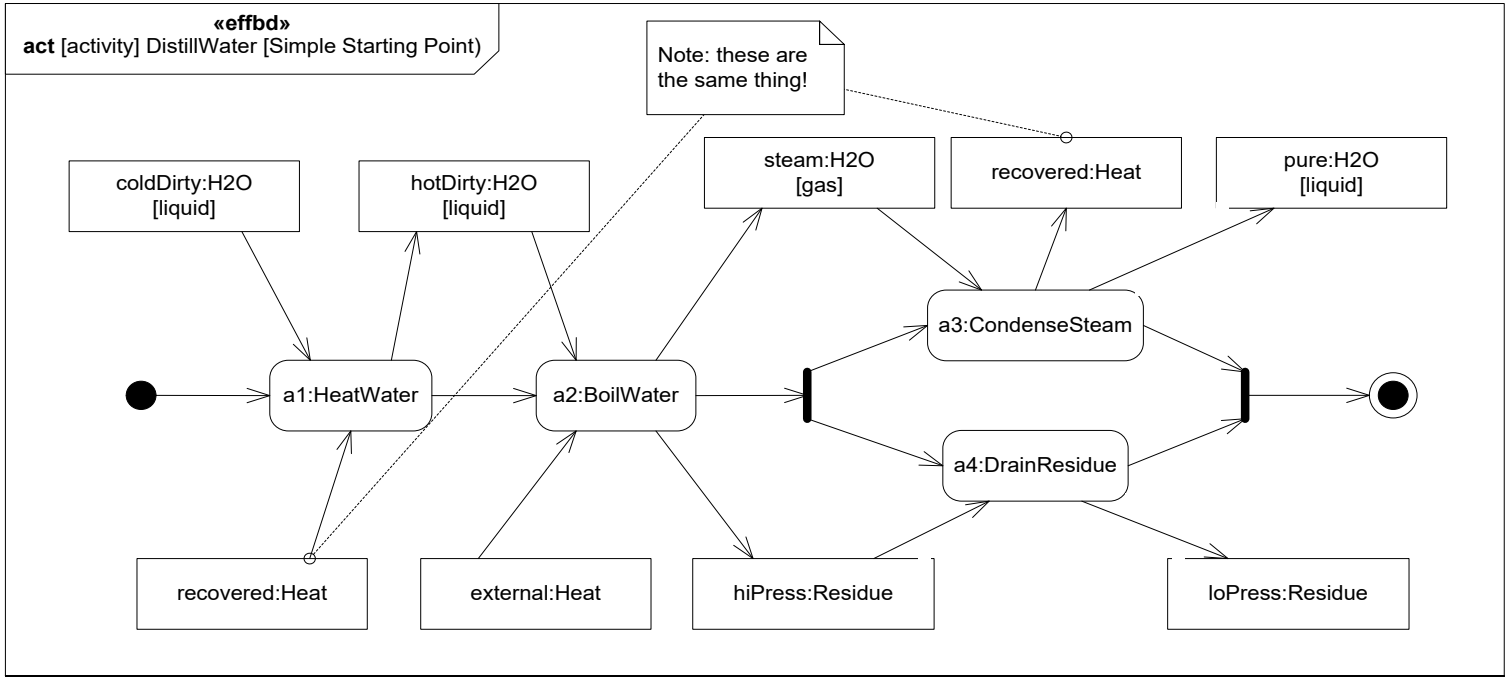
Evaluating Measures of Effectiveness



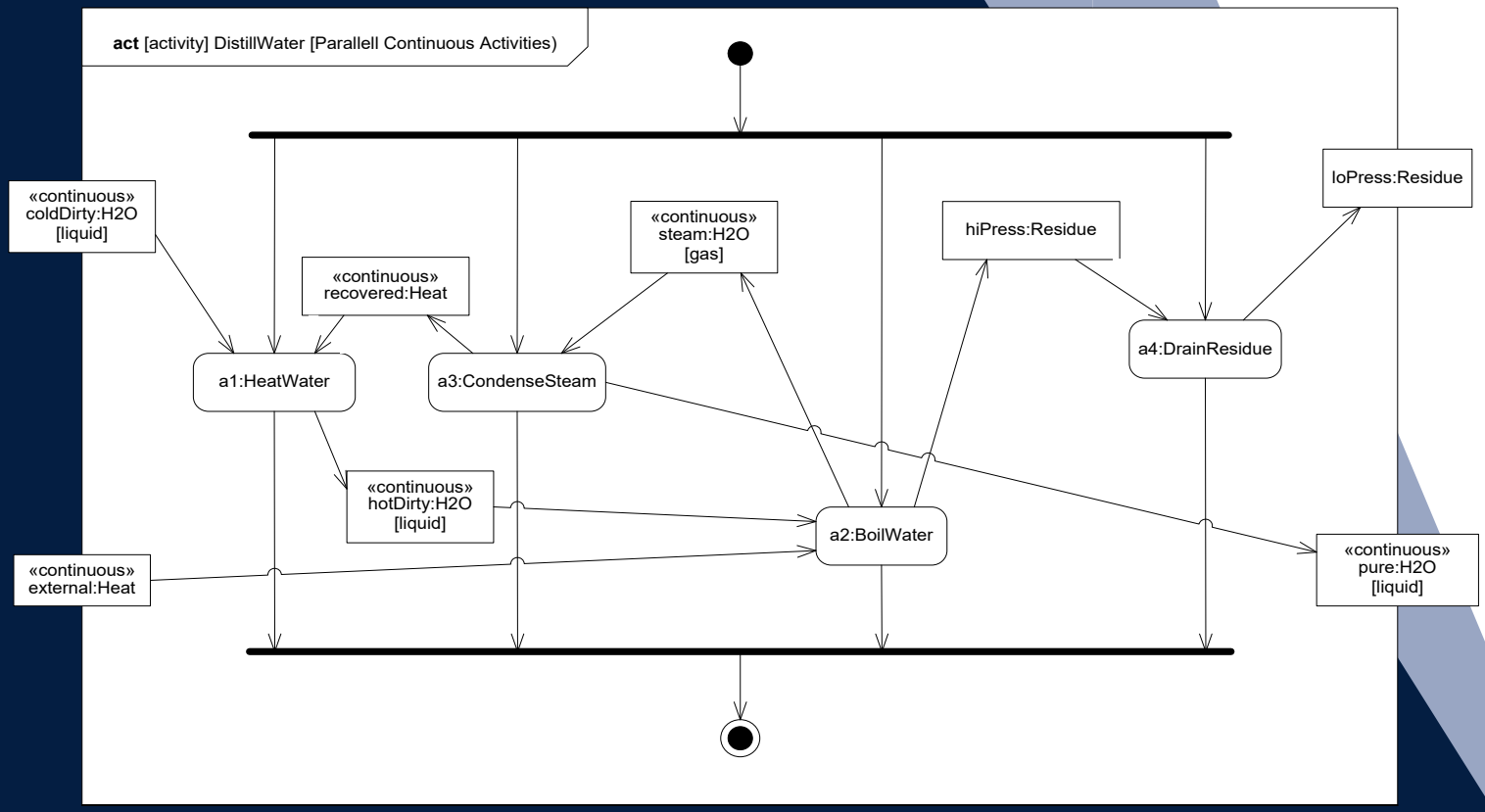
Distiller Example



Distill Water Activity Diagram (Initial)



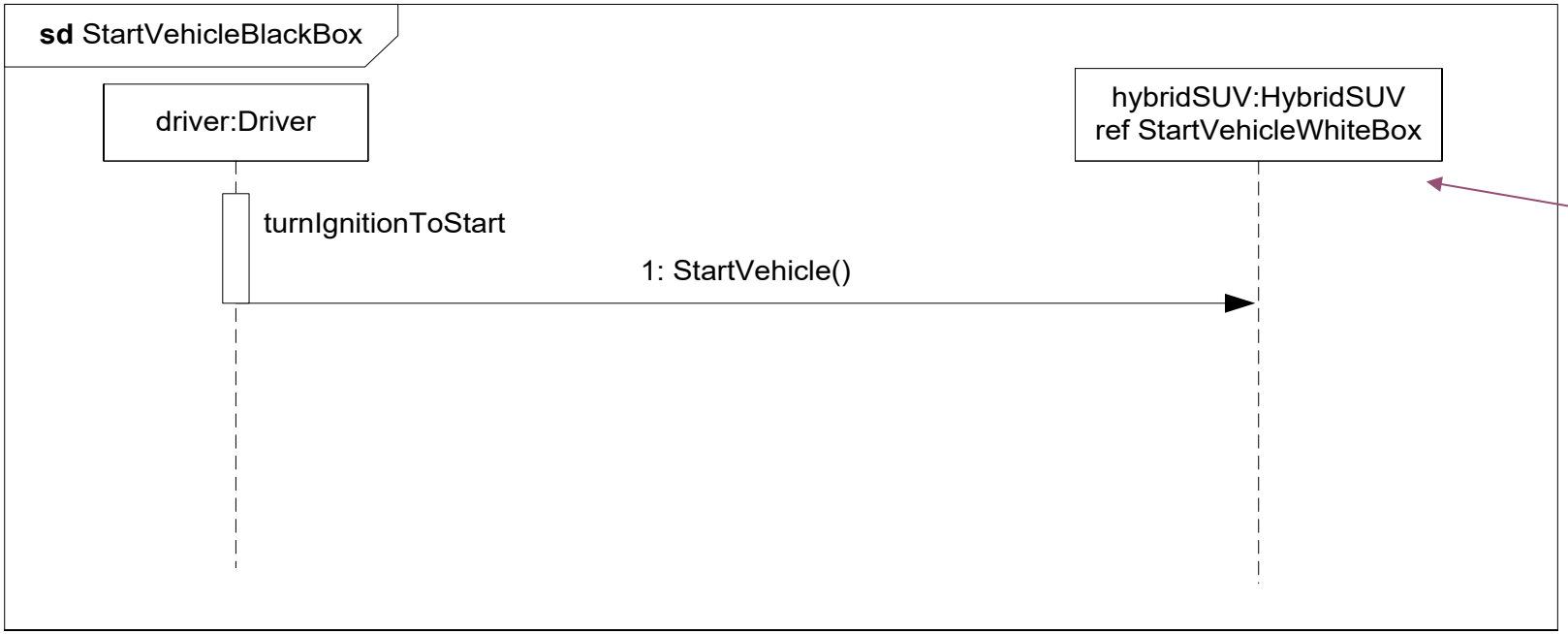
Distill Water Activity Diagram (Continuous Flow Modeling)



Interactions

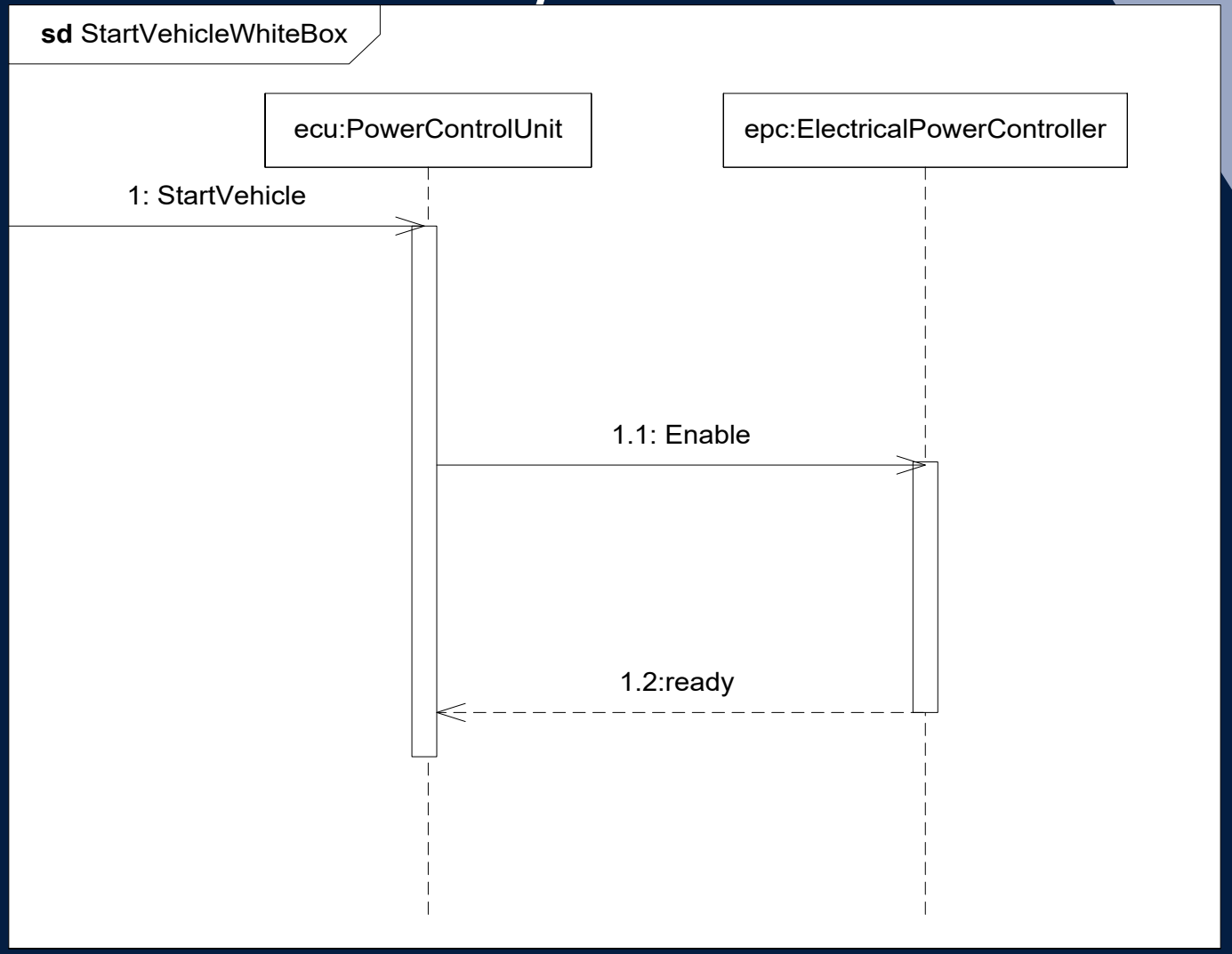
- Sequence diagrams provide representations for message based behavior
  - Represents flow of control
    - Less effective than activities for representing inputs from multiple sources
  - UML 2 sequence diagrams significantly more scalable by providing reference sequences, control logic, and lifeline decomposition
- Timing diagrams provide representations for typical system timelines and value properties vs time
- No change to UML
  - Minor clarification on continuous time representations

Black Box Sequence (StartVehicle)



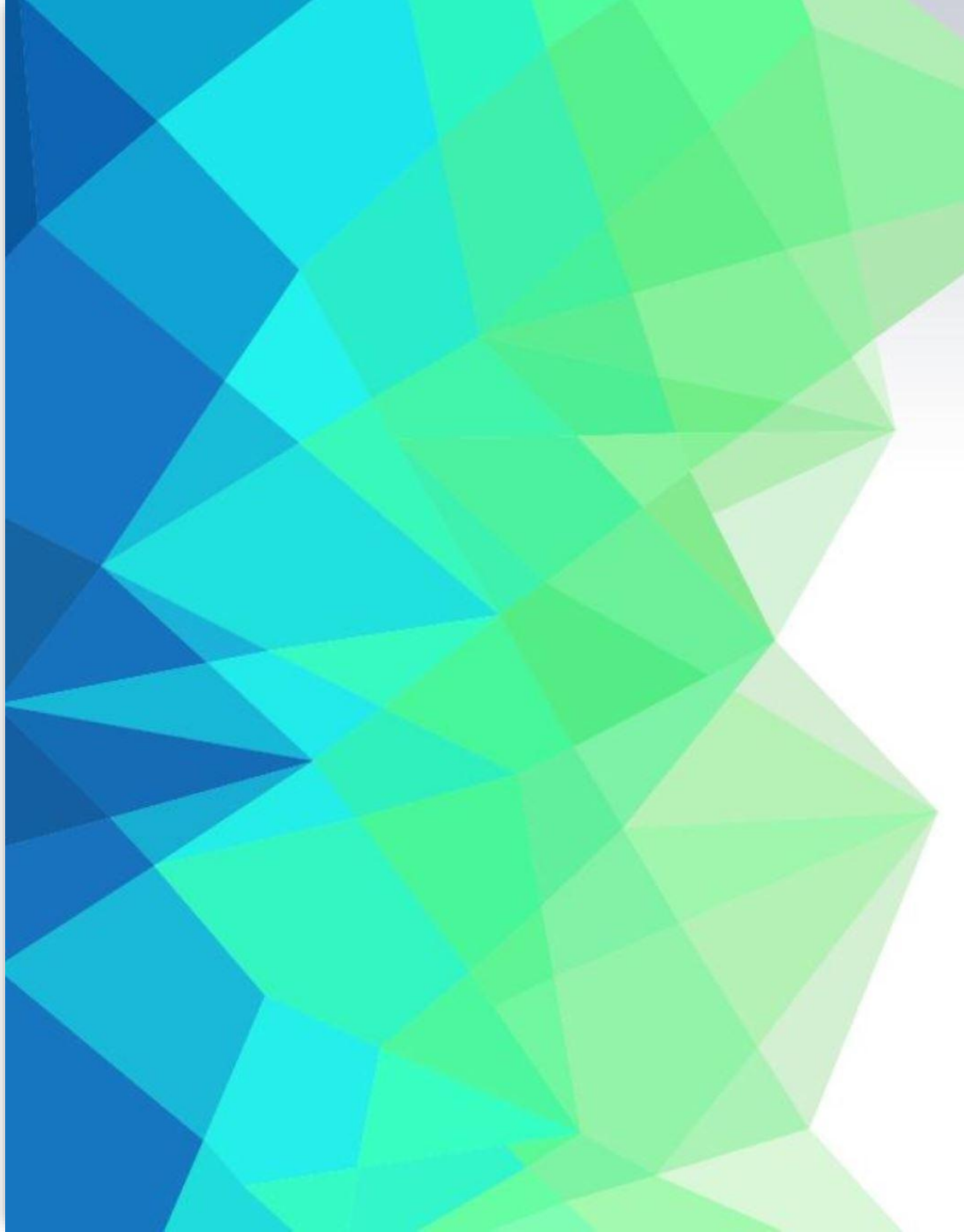
References Lifeline Decomp
For White Box Interaction

White Box Sequence (StartVehicle)



Requirements

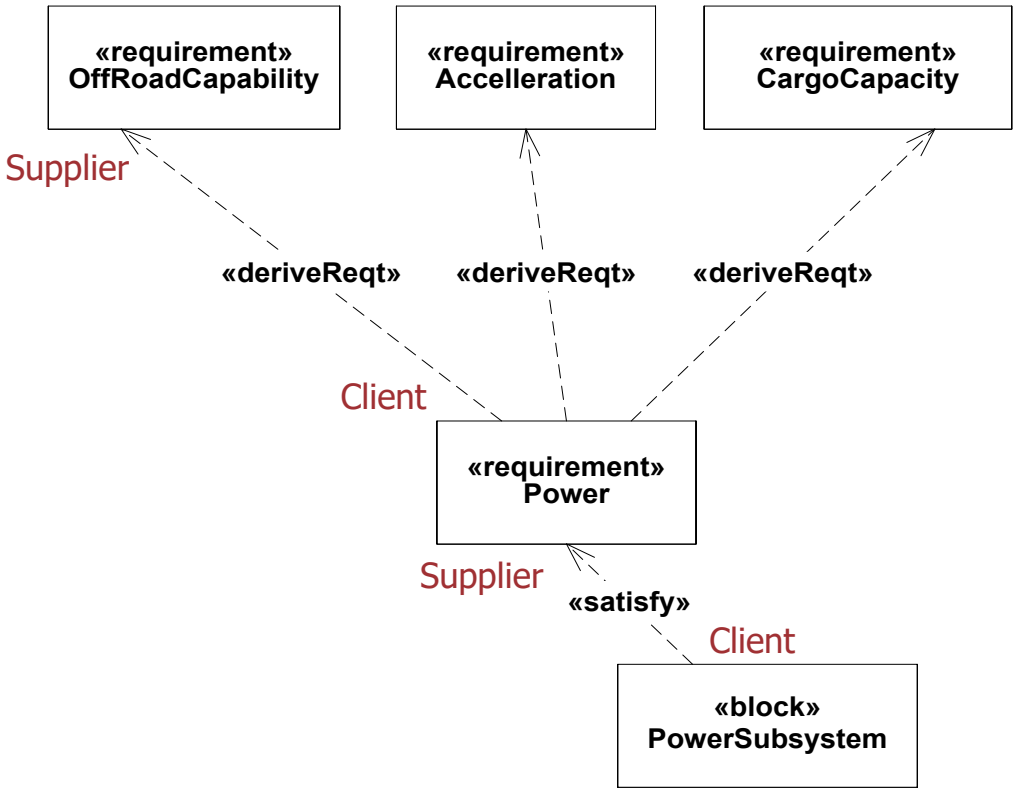
- Requirements represents a text based requirement
  - Minimal properties specified for id and text based on user feedback
  - Stereotype mechanism used to categorize requirements (e.g. functional, physical)
    - Able to specify constraints on what design elements can satisfy the requirement (refer to Appendix C.2)
  - Stereotype of class (abstract) without instances
- Requirements containment used to specify requirements hierarchy as a collection of requirements (e.g., a specification)
    - SST uses cross hairs notation vs black diamond composition to be consistent with containment semantics
- Requirements relationships based on subclasses of dependency
  - Derive, Satisfy, Verify, Refine, ..



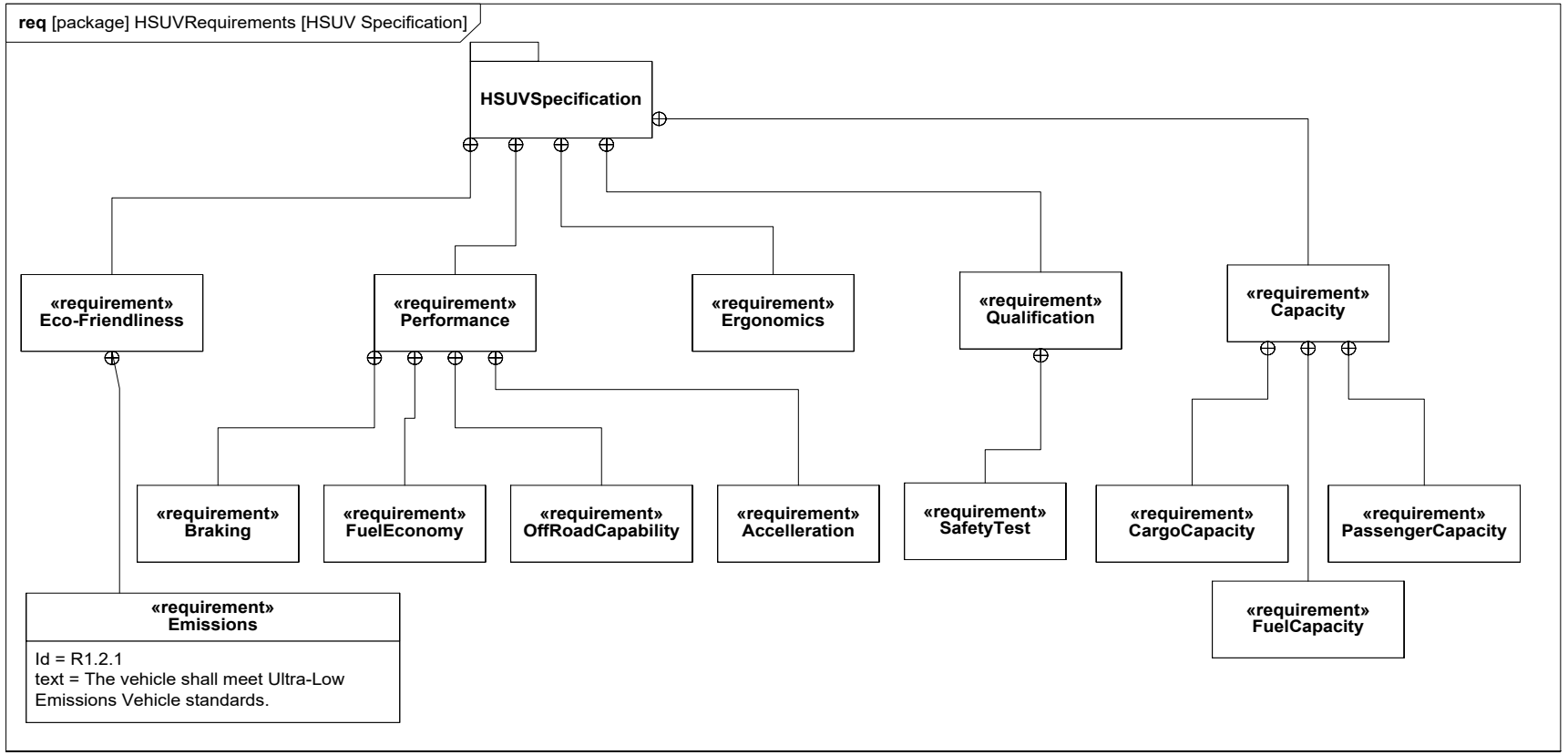
Dependencies

- Used to specify relationships among requirements (other uses as well)
  - Different concept for SE's with arrow direction reversed from typical requirements flow-down
    - Refer to next slide
- Represents a relationship between client and supplier elements
  - Client depends on supplier
    - A change in supplier results in a change in client
    - Application to requirements: A change in requirement (supplier) results in a change in design element that satisfies it (client) or requirement derived from it (client)

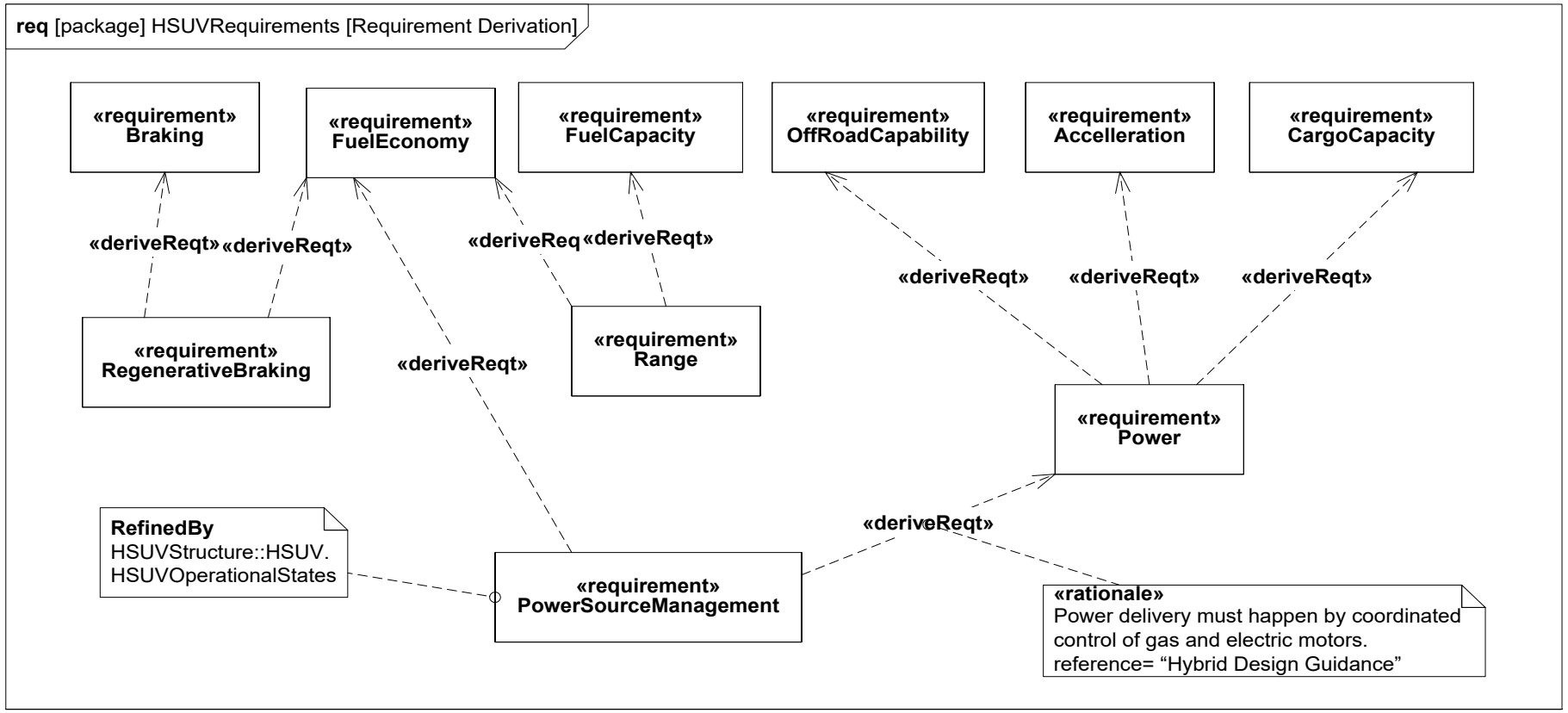
Example of Derive/Satisfy Requirement Dependencies

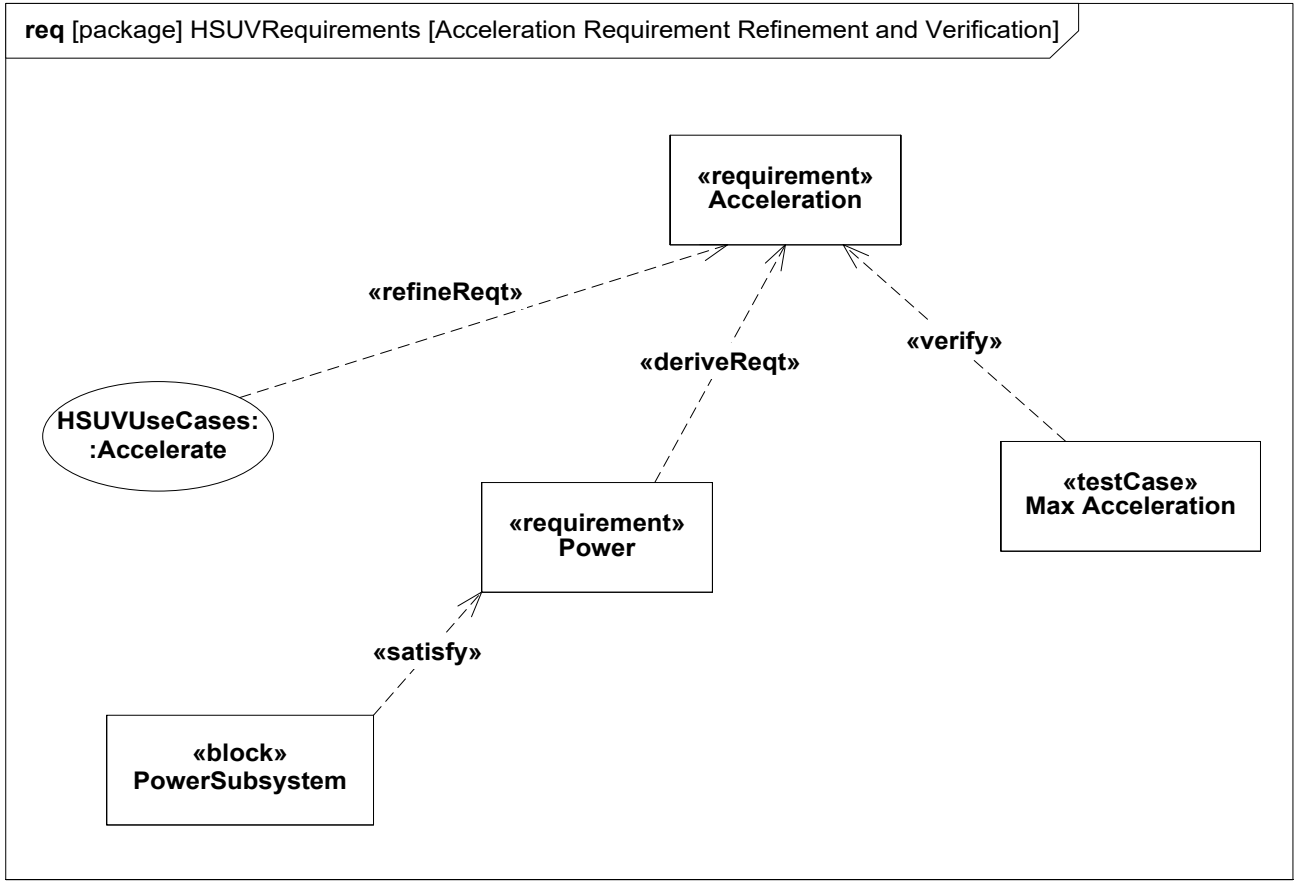


Requirements Breakdown



Requirements Derivation





Reqts Refinement/Verification

table [requirement] Capacity [Decomposition of Capacity Requirement]

id	name	text
4	Capacity	The Hybrid SUV shall carry 5 adult passengers, along with sufficient luggage and fuel for a typical weekend campout.
4.1	CargoCapacity	The Hybrid SUV shall carry sufficient luggage for 5 people for a typical weekend campout.
4.2	FuelCapacity	The Hybrid SUV shall carry sufficient fuel for a typical weekend campout.
4.3	PassengerCapacity	The Hybrid SUV shall carry 5 adult passengers.

table [requirement] Performance [Decomposition of Performance Requirement]

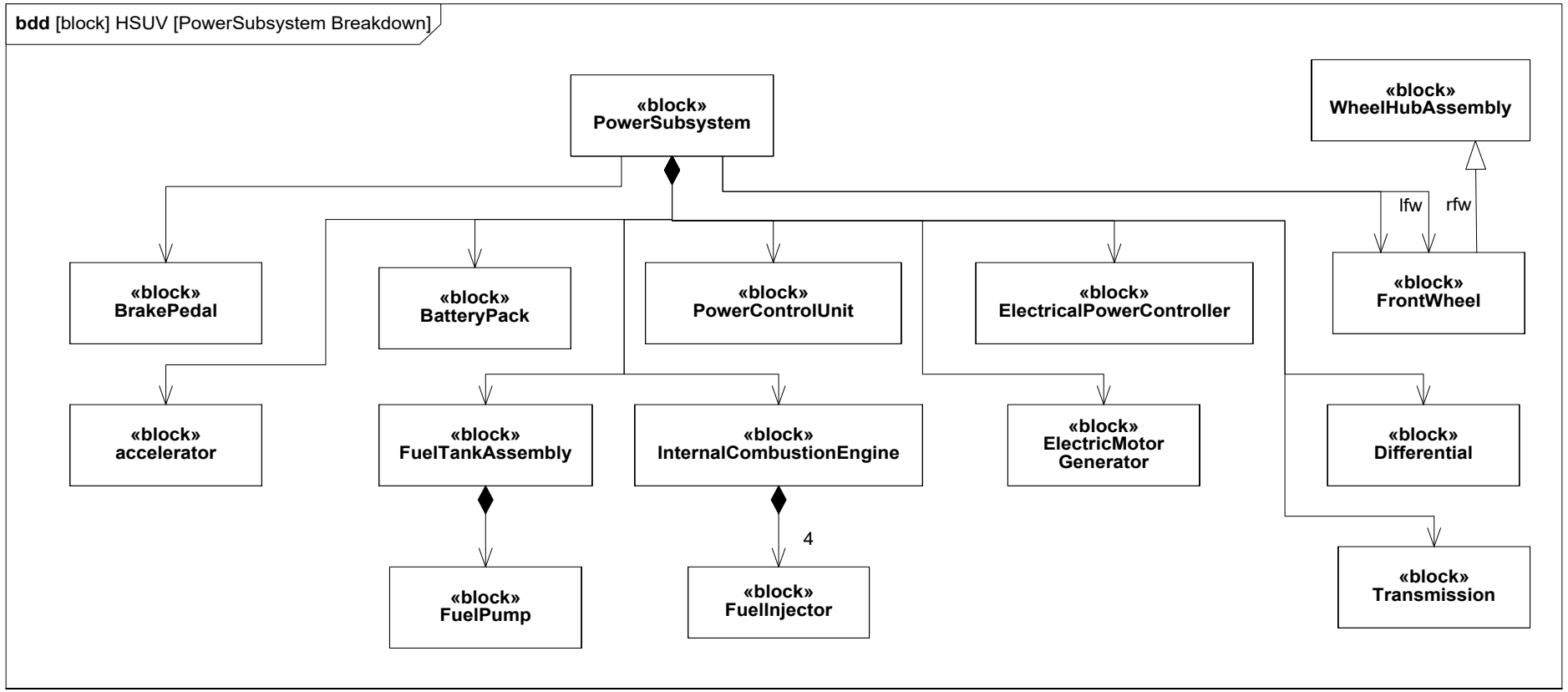
id	name	text
2	Performance	The Hybrid SUV shall have the braking, acceleration, and off-road capability of a typical SUV, but have dramatically better fuel economy.
2.1	Braking	The Hybrid SUV shall have the braking capability of a typical SUV.
2.2	FuelEconomy	The Hybrid SUV shall have dramatically better fuel economy than a typical SUV.
2.3	OffRoadCapability	The Hybrid SUV shall have the off-road capability of a typical SUV.
2.4	Acceleration	The Hybrid SUV shall have the acceleration of a typical SUV.

table [requirement] Performance [Tree of Performance Requirements]

id	name	relation	id	name	relation	id	name
2.1	Braking	deriveReqt	d.1	RegenerativeBraking			
2.2	FuelEconomy	deriveReqt	d.1	RegenerativeBraking			
2.2	FuelEconomy	deriveReqt	d.2	Range			
4.2	FuelCapacity	deriveReqt	d.2	Range			
2.3	OffRoadCapability	deriveReqt	d.4	Power	deriveReqt	d.2	PowerSourceManagement
2.4	Acceleration	deriveReqt	d.4	Power	deriveReqt	d.2	PowerSourceManagement
4.1	CargoCapacity	deriveReqt	d.4	Power	deriveReqt	d.2	PowerSourceManagement

Requirements Tables & Trees

Power Subsystem Breakdown



Power Subsystem IBD