DASSAULT SYSTEMES

# CATIA USER SYMPOSIUM AMERICAS 2026

MAY 18-21 | RENO | NEVADA | USA

# AI-Assisted MBSE with SysML v2 in Cameo 2026x

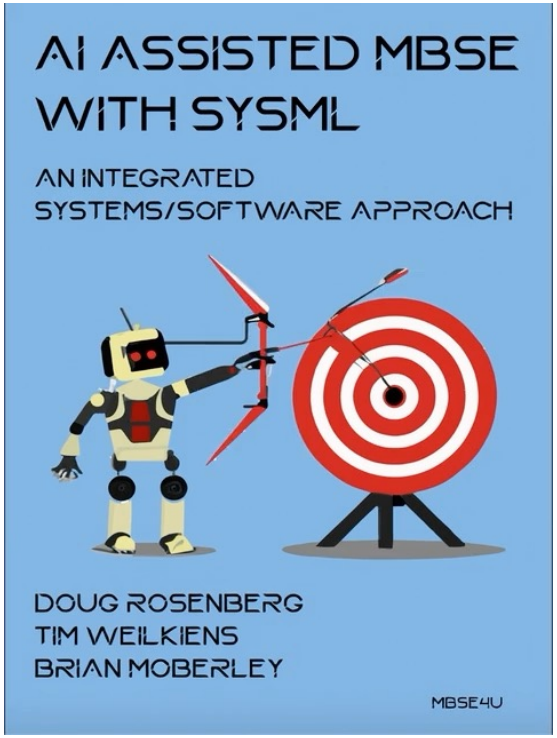
## A Template-Driven Path to Radical Productivity Improvement

*Doug Rosenberg – Parallel Agile, Inc. and Caltech CTME*

# AIM has moved from concept to reality

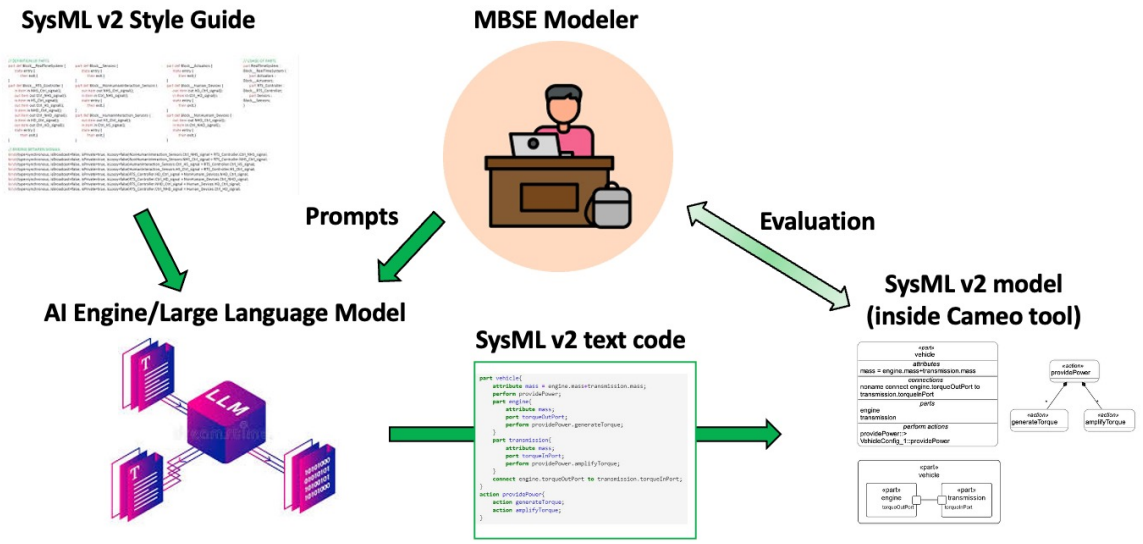
AI Assisted MBSE (AIM) is now a reality with the introduction of SysML v2 support in Cameo 2026x

AIM 2nd Edition is now being written, as the current edition was published before SysML v2 was available



# Use AI to generate SysML v2 textual notation

- We explored using AI as a Subject Matter Expert (#AISME) in the AI Assisted MBSE book using an Electron Microscope
- Now, we can leverage that capability by “code generating” SysML v2 directly from AI
- Cameo 2026x can import this v2 textual notation and create SysML diagrams from it



# What is the SysML v2 Style Guide and why do we need it?

- **Cameo-compliant SysML v2 code samples** for use as syntax reference with AI Code Generation
- Needed because code samples for SysML v2 are not available in quantities compared to languages like JavaScript
- Fortunately, AI is good at “follow this pattern”
- There are 9 SysML diagram types (v1), so we need a set of at least 9 code samples to match v1 capability using v2
- These samples guide AI in generating syntactically valid v2 code

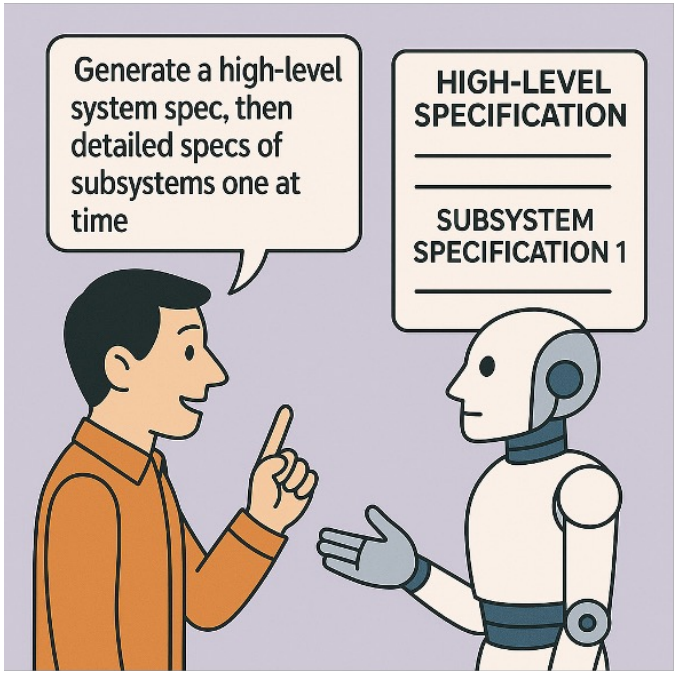
```
part def DescentEngine {
    action ignite : Ignite;
    action adjustThrust : AdjustThrust;
    action shutdown : Shutdown;
}
part def DescentFuelTanks {
    action supplyFuel : SupplyFuel;
    action monitorFuelLevel : MonitorFuelLevel;
}
part def GuidanceComputer {
    action descendToLunarSurface : DescendToLunarSurface;
    action ascendFromLunarSurface : AscendFromLunarSurface;
    action dockWithCommandModule : DockWithCommandModule;
    action undockFromCommandModule : UndockFromCommandModule;
    action adjustCourse : AdjustCourse_Guidance;
}
part def TelemetrySubsystem {
    action transmitData : TransmitData;
    action receiveCommands : ReceiveCommands;
}
part def ManeuveringSubsystem {
    action fireRCSThrusters : FireRCSThrusters;
```

# There was a significant learning curve

- We started with Cameo 2024x r3 beta
- **THANK YOU** to Osvaldas Jankauskas from Dassault for helping me get started with v2
- Debugging SysML v2 code with errors is not a lot of fun
- Note that SysML v2 compliant is not identical to Cameo-compliant
- We started teaching AIM at Caltech CTME and found success when providing students with code samples

## As we gained experience, we got more ambitious

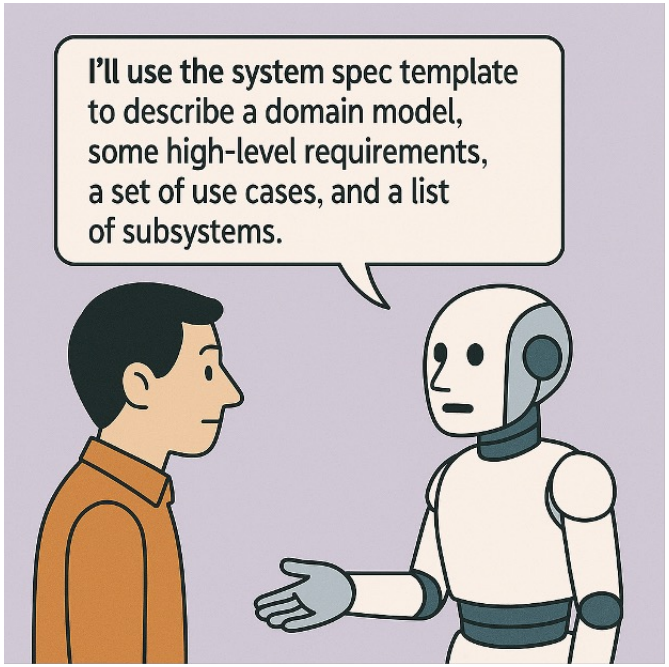
- Once our “right the first time” rate exceeded 90% we decided to try combining templates
- We tried a “system spec” template which combined Requirements, Use Cases, Domain Model, and Subsystems
- Then we tried a ”logical architecture” template which included Parts, State Machine, Requirements and IBD, suitable for a subsystem at a time
- Template contents can be tailored as desired
- Currently we are investigating SysML as spec for agentic AI code generators



# We used our System Spec template to generate v2 code for a lunar lander

**Lunar Lander V2 Model**

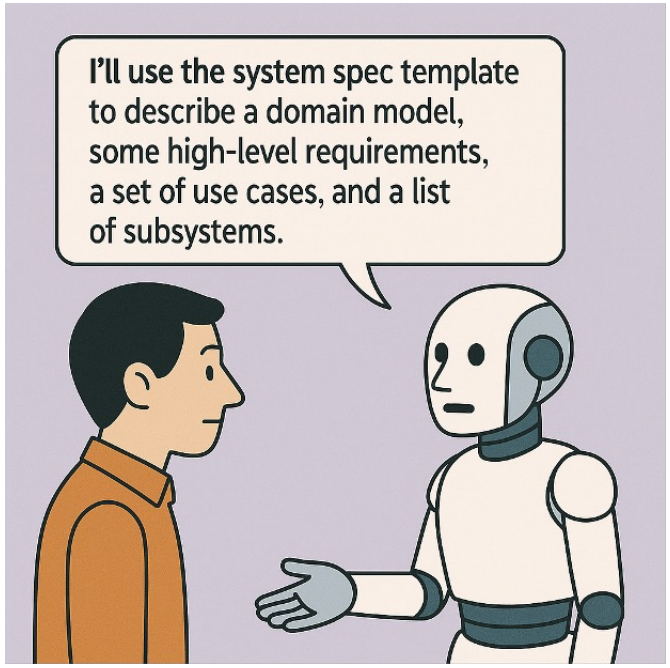
1. Domain Model
2. Glossary
3. Top-Level Requirements
4. Top-Level State Machine
5. Top-Level Subsystem Decomposition
6. Ascent Stage Subsystem
   - Part Decomposition
   - Requirements
   - State Machine
   - Connections
7. Descent Stage Subsystem
   - Part Decomposition
   - Requirements
   - State Machine
   - Connections




*The complete example is published as Appendix B of Pain-Free MBSE*

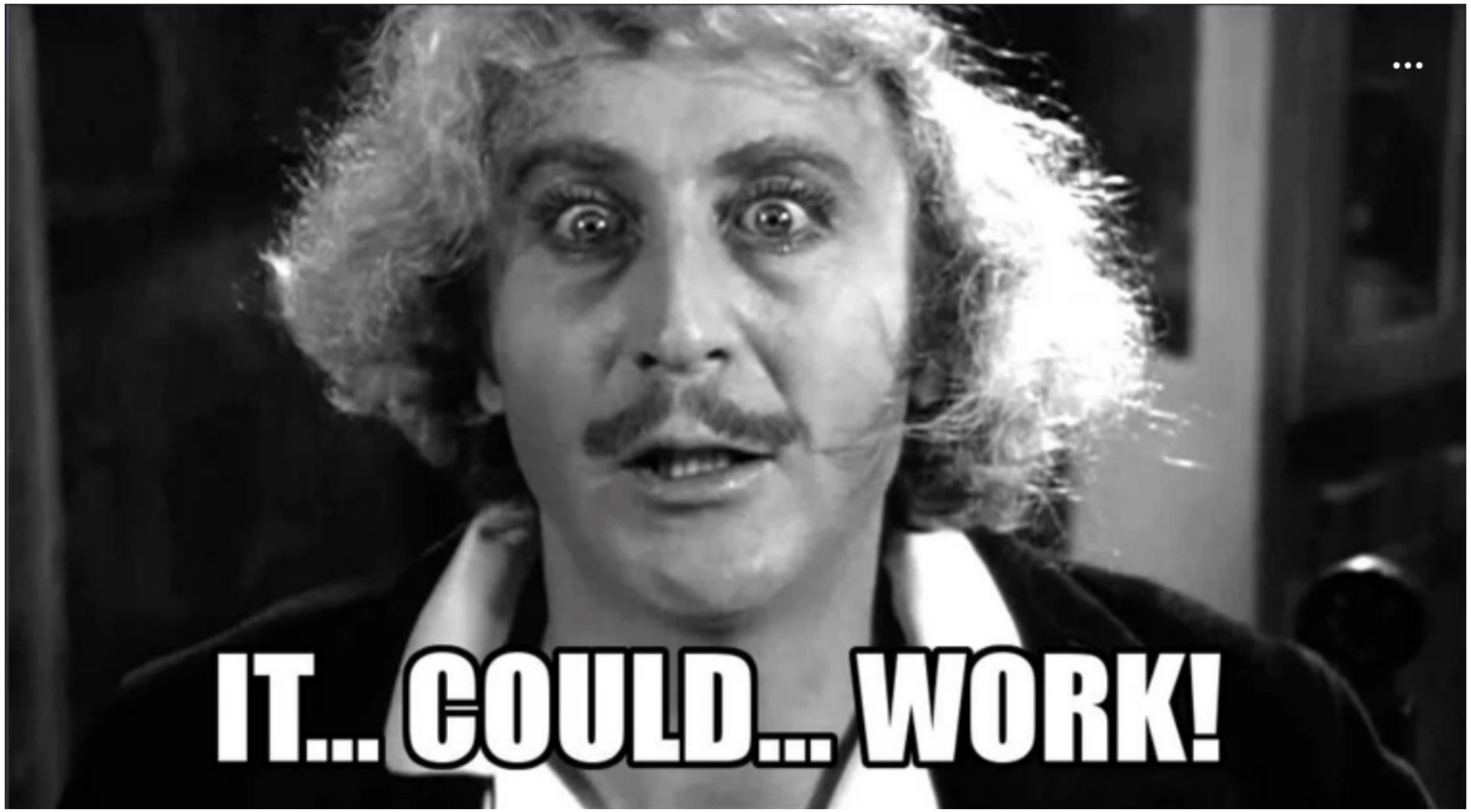
# One prompt to generate the System Spec

*Using the System Spec template for reference, generate a high-level spec for a Lunar Lander*




*The complete example is published as Appendix B of Pain-Free MBSE*

# One prompt for a high-level spec???



# Lunar Lander Domain Model

```
package 'Lunar Lander V2 Model' {
    package LunarLanderDomain_Grouped {
        ////////////////////////////////
        // Top-level conceptual groupings
        ////////////////////////////////
        part def LunarLander {
            part ascent         : Ascent;
            part descent        : Descent;
            part communications : Communications;
            part docking        : Docking;
            part controls       : Controls;
            part fuelSystem     : FuelSystem;
            part crewCabin      : CrewCabin;
        }

        // ---- Group: Ascent ----
        part def Ascent {
            part ascentEngine            : Engine;
            part reactionControlThrusters : ThrusterSet;
        }

        // ---- Group: Descent ----
        part def Descent {
            part descentEngine  : Engine;
            part landingGear    : LandingGear;
            part footpads       : FootpadSet;
            part contactProbes  : ContactProbeSet;
            part radarAltimeter : RadarAltimeter;
        }

        // ---- Group: Communications ----
        part def Communications {
            part antenna       : Antenna;
            part rfTransceiver : RFTransceiver;
        }

```

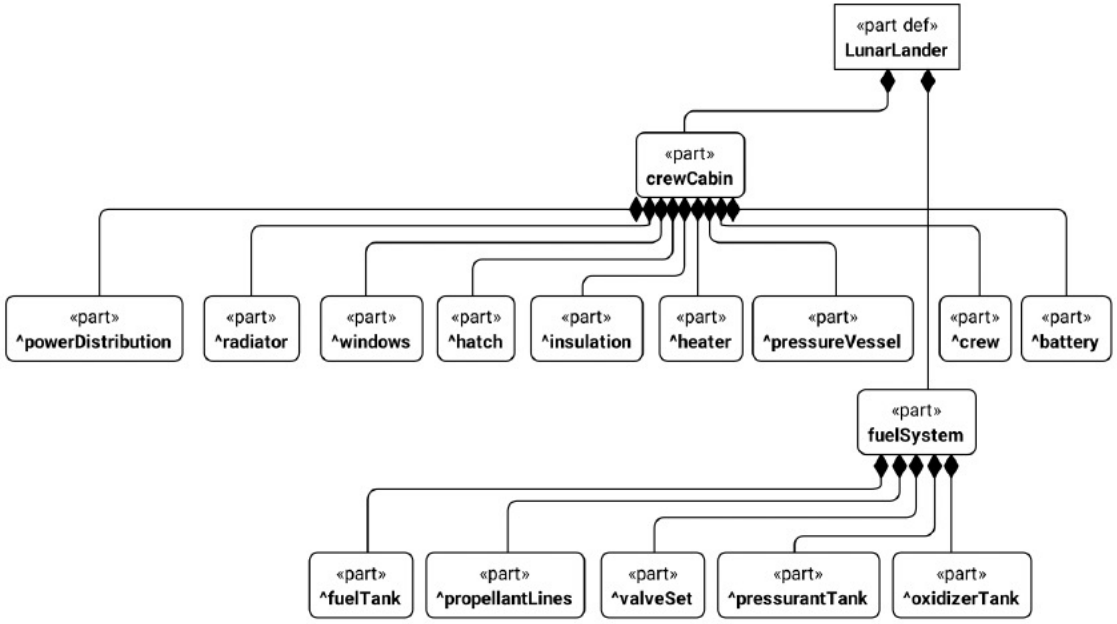


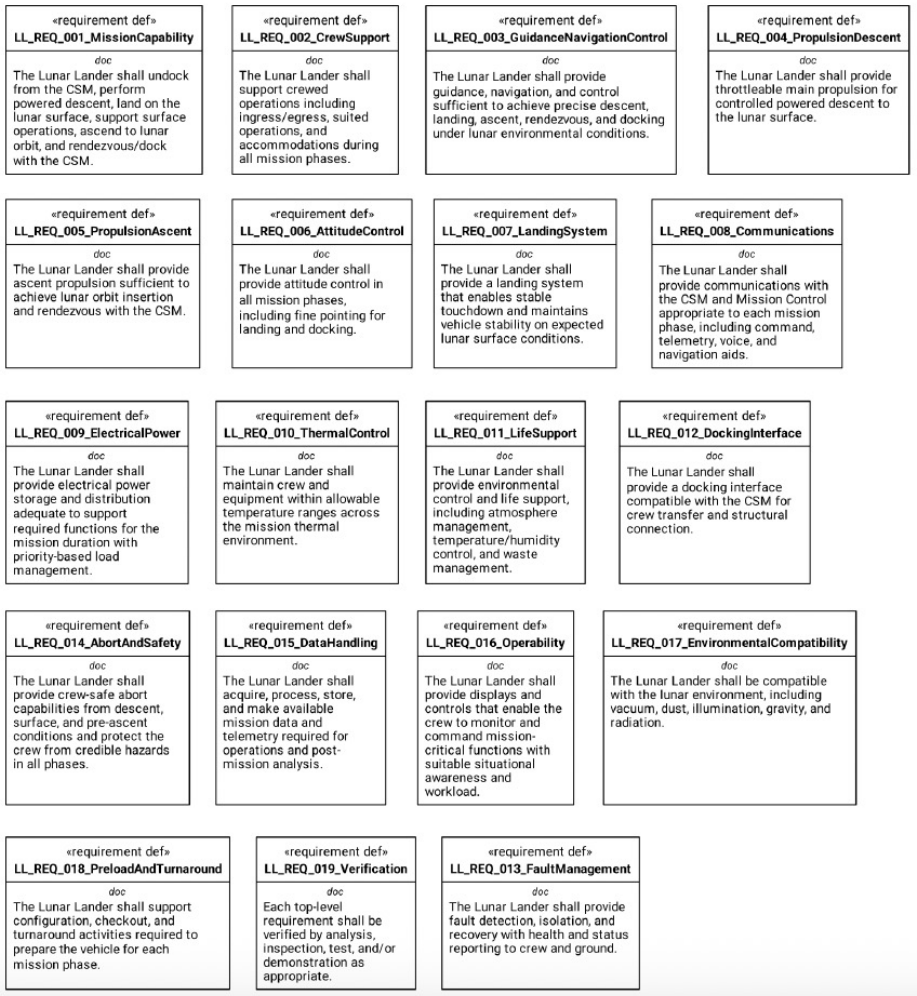

**Figure 8. Lunar Lander Domain Model (General View)**

# AI produces a Glossary with definitions

| Term | Description |
|---|---|
| Antenna | Transmits and receives radio signals between the lander, the CSM, and Earth. |
| AscentEngine | Provides main thrust for lunar ascent and orbital insertion. |
| Battery | Stores electrical energy for vehicle power. |
| CommandServiceModule | Companion spacecraft in lunar orbit used for docking and crew transfer. |
| ContactProbes | Detect surface contact just before landing. |
| Crew | Astronauts operating the lunar lander during mission phases. |
| DescentEngine | Provides throttleable thrust for powered descent and landing. |
| DisplaysAndControls | Present vehicle status and accept crew input for manual control. |

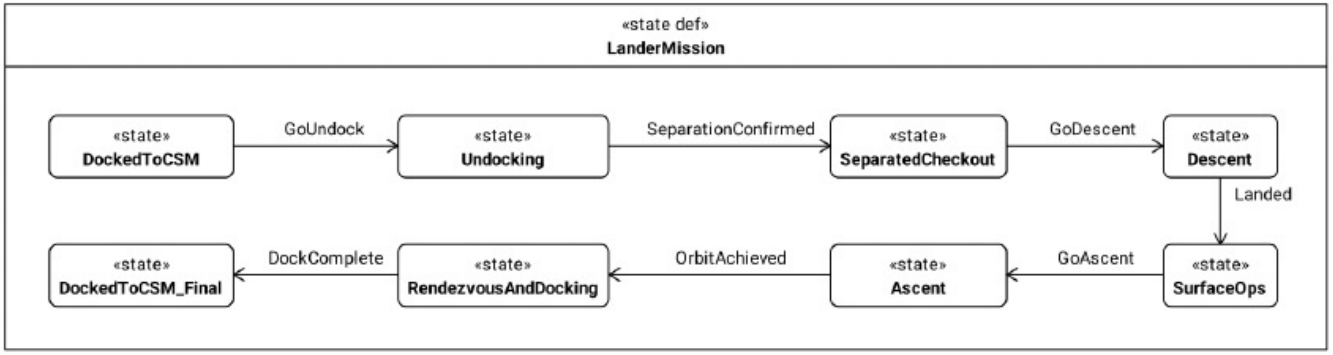
# Top Level System Requirements

```
package 'Lunar Lander V2 Model' {
    package LunarLanderRequirements {
        requirement def LL_REQ_001_MissionCapability {
            doc /* The Lunar Lander shall undock from the CSM, perform powered
                 * descent, land on the lunar surface, support surface operations,
                 * ascend to lunar orbit, and rendezvous/dock with the CSM. */
        }
        requirement def LL_REQ_002_CrewSupport {
            doc /* The Lunar Lander shall support crewed operations including
                 * ingress/egress, suited operations, and accommodations during
                 * all mission phases. */
        }
        requirement def LL_REQ_003_GuidanceNavigationControl {
            doc /* The Lunar Lander shall provide guidance, navigation, and control
                 * sufficient to achieve precise descent, landing, ascent, rendezvous,
                 * and docking under lunar environmental conditions. */
        }
        requirement def LL_REQ_004_PropulsionDescent {
            doc /* The Lunar Lander shall provide throttleable main propulsion for
                 * controlled powered descent to the lunar surface. */
        }
        requirement def LL_REQ_005_PropulsionAscent {
            doc /* The Lunar Lander shall provide ascent propulsion sufficient to
                 * achieve lunar orbit insertion and rendezvous with the CSM. */
        }
        requirement def LL_REQ_006_AttitudeControl {
            doc /* The Lunar Lander shall provide attitude control in all mission
                 * phases, including fine pointing for landing and docking. */
        }
        requirement def LL_REQ_007_LandingSystem {
            doc /* The Lunar Lander shall provide a landing system that enables
                 * stable touchdown and maintains vehicle stability on expected
                 * lunar surface conditions. */
        }
```



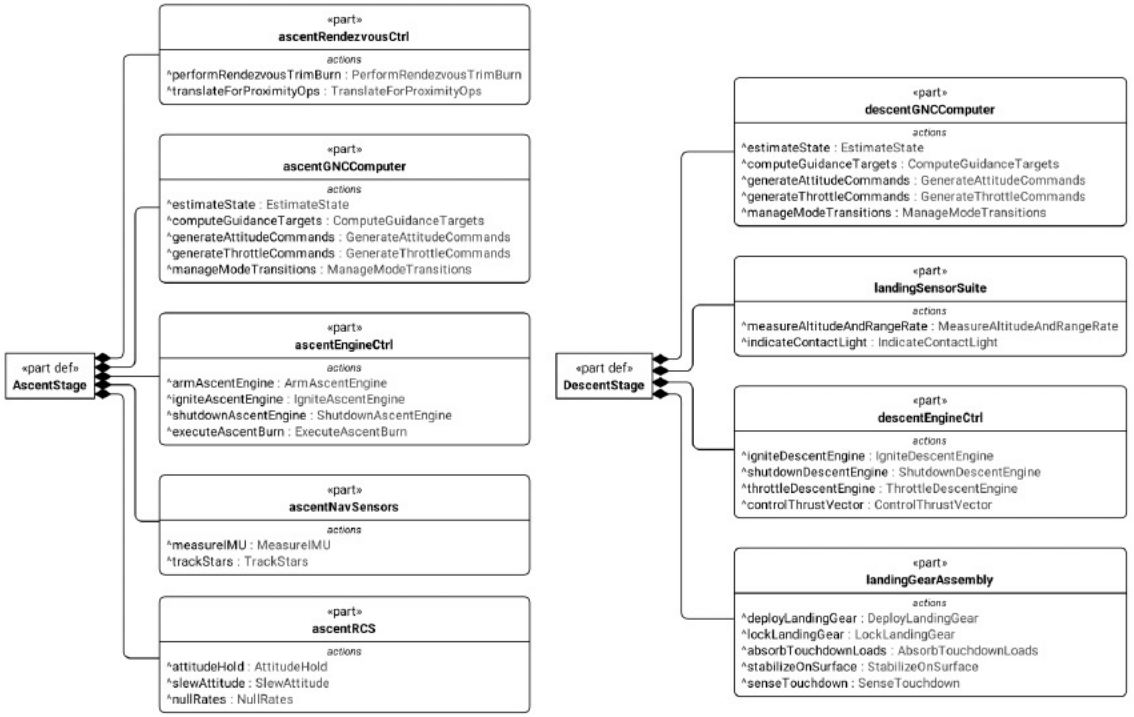
# Top Level State Machine

```
package 'Lunar Lander V2 Model' {
    package LunarLanderTopLevelStateMachine {
        item def GoUndock;
        item def SeparationConfirmed;
        item def GoDescent;
        item def Landed;
        item def GoAscent;
        item def OrbitAchieved;
        item def InProximity;
        item def DockComplete;
        part def LunarLander {
            exhibit state LanderMission;
        }
        state def LanderMission {
            state DockedToCSM;
            state Undocking;
            state SeparatedCheckout;
            state Descent;
            state SurfaceOps;
            state Ascent;
            state RendezvousAndDocking;
            state DockedToCSM_Final;
            transition first entryAction then DockedToCSM;
            transition t1 first DockedToCSM accept GoUndock then Undocking;
            transition t2 first Undocking accept SeparationConfirmed then
            ↪ SeparatedCheckout;
            transition t3 first SeparatedCheckout accept GoDescent then Descent;
            transition t4 first Descent accept Landed then SurfaceOps;
            transition t5 first SurfaceOps accept GoAscent then Ascent;
            transition t6 first Ascent accept OrbitAchieved then RendezvousAndDocking;
            transition t7 first RendezvousAndDocking accept DockComplete then
            ↪ DockedToCSM_Final;
        }
    }
}
```



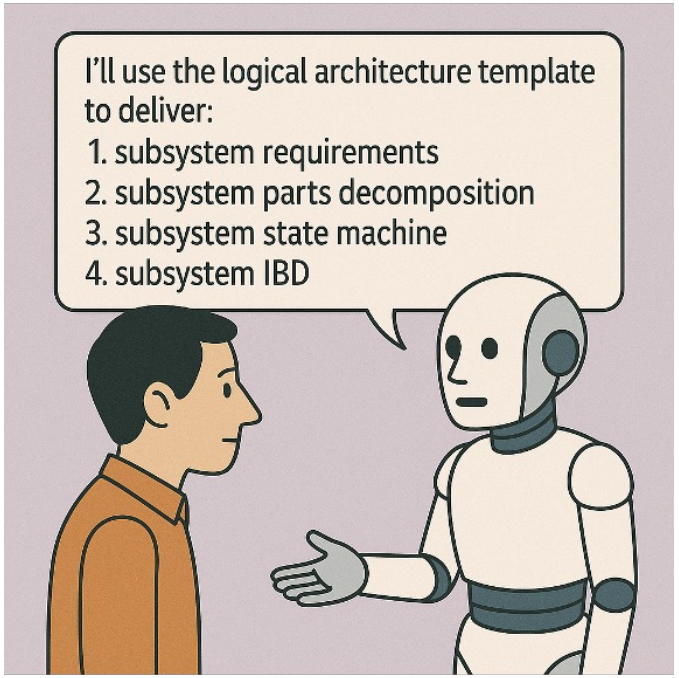
# Top Level Subsystems

```
70          part def AscentStage {
71              part ascentEngineCtrl : AscentEngineControl;
72              part ascentRendezvousCtrl : RendezvousControl;
73              part ascentRCS : RCSModule;
74              part ascentGNCComputer : GNCComputer;
75              part ascentNavSensors : NavSensorSuite;
76          }
77          part def DescentStage {
78              part descentEngineCtrl : DescentEngineControl;
79              part landingSensorSuite : LandingSensorSuite;
80              part landingGearAssembly : LandingGearAssembly;
81              part descentGNCComputer : GNCComputer;
82          }
83          part def CrewCabin {
84              part eclssAir : ECLSS_AirManagement;
85              part eclssWaterWaste : ECLSS_WaterWaste;
86              part crewInterface : CrewInterface;
87              part cabinThermal : ThermalLoop;
88              part cabinPowerPanel : PowerDistributionPanel;
89              part crewProtection : CrewProtection;
90          }
91          part def CommunicationsAndTelemetry {
92              part rfTransceiverUnit : RFTransceiverUnit;
93              part commsAntennaSystem : AntennaSystem;
94              part telemetryRecorder : TelemetryRecorder;
```

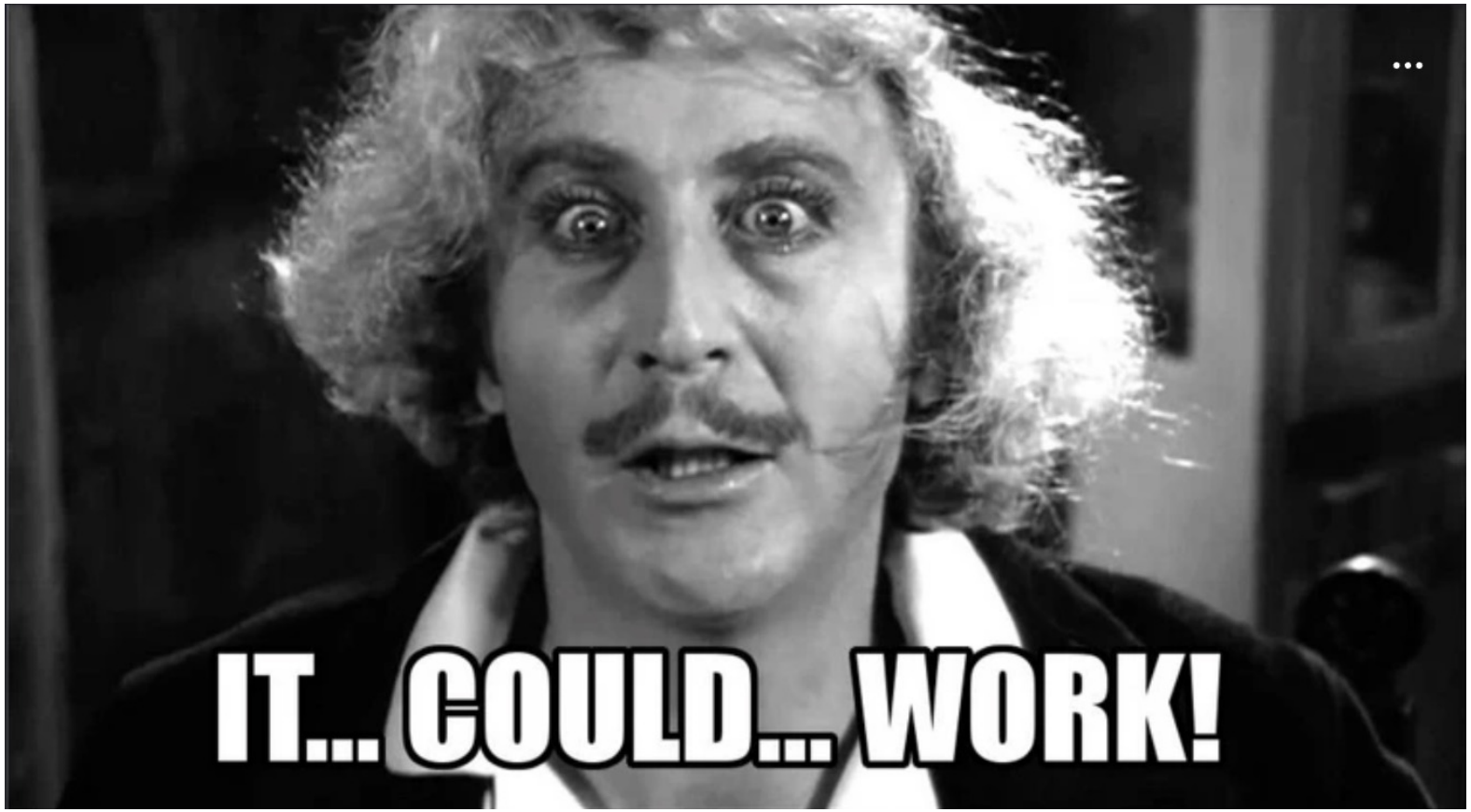


## Generate Logical Architecture specs a Subsystem at a time

*Using the Logical Architecture template for reference, generate a Subsystem spec for the Ascent Subsystem of a Lunar Lander*

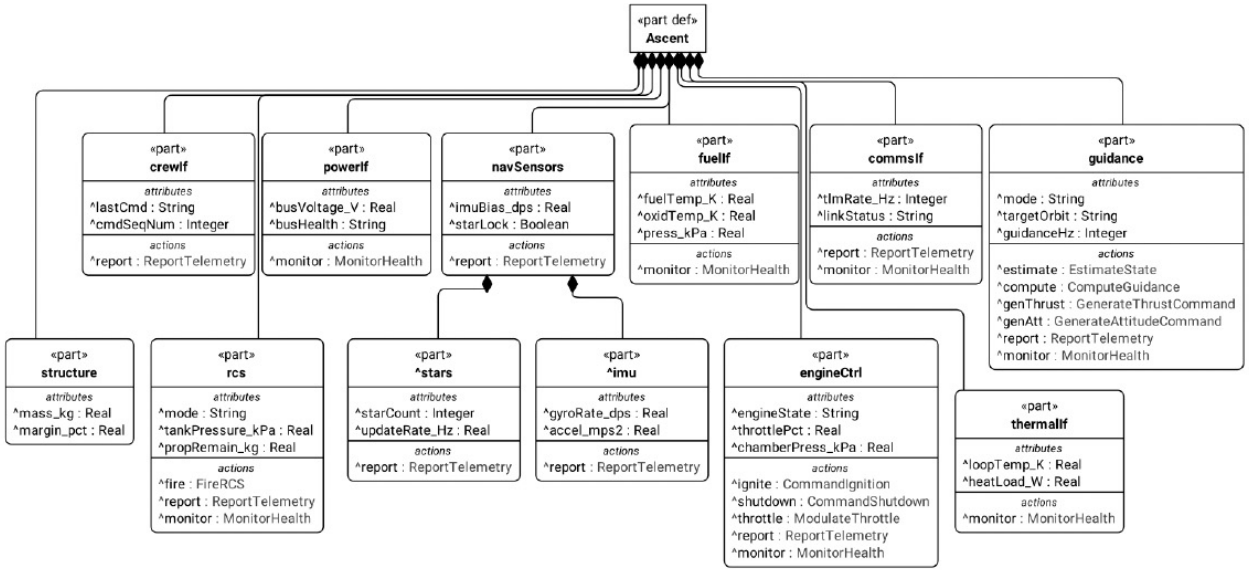


# One prompt per subsystem???



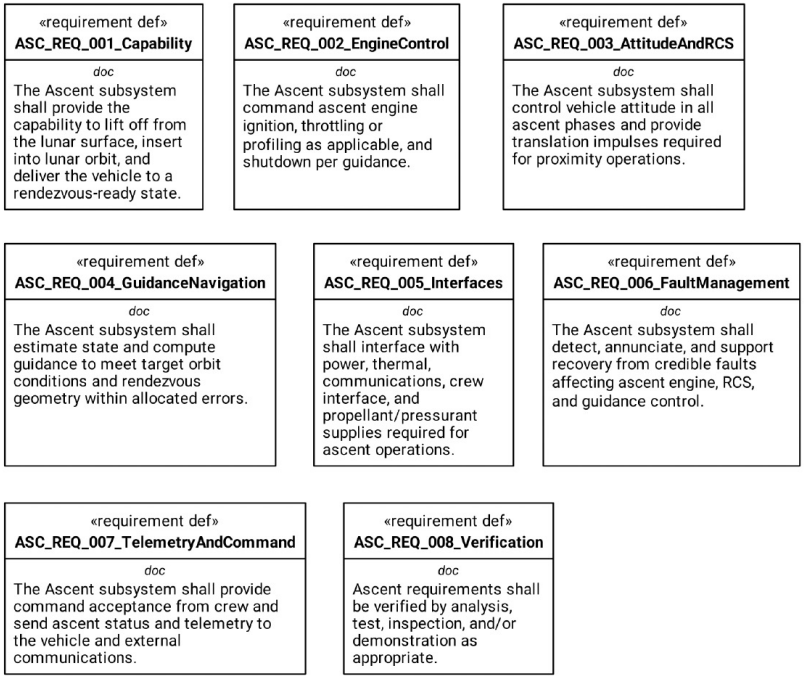
# Ascent Stage Part Decomposition

```
package 'Lunar Lander V2 Model' {
    package LunarLanderSubsystemsDecomposition {
        package Ascent_Stage {
            package PartDecomposition {
                private import ScalarValues::**;
                private import SI::**;
                action def ComputeGuidance;
                action def EstimateState;
                action def GenerateThrustCommand;
                action def GenerateAttitudeCommand;
                action def CommandIgnition;
                action def CommandShutdown;
                action def ModulateThrottle;
                action def FireRCS;
                action def MonitorHealth;
                action def ReportTelemetry;
                part def AscentEngineControl {
                    attribute engineState : String;
                    attribute throttlePct : Real;
                    attribute chamberPress_kPa : Real;
                    action ignite : CommandIgnition;
                    action shutdown : CommandShutdown;
                    action throttle : ModulateThrottle;
                    action report : ReportTelemetry;
                    action monitor : MonitorHealth;
                }
                part def RCSModule {
                    attribute mode : String;
                    attribute tankPressure_kPa : Real;
                    attribute propRemain_kg : Real;
                    action fire : FireRCS;
                    action report : ReportTelemetry;
                    action monitor : MonitorHealth;
                }
                part def NavSensorSuite {
                    attribute imuBias_dps : Real;
                    attribute starLock : Boolean;
                    action report : ReportTelemetry;
                    part imu : IMU;
                    part stars : StarTracker;
```



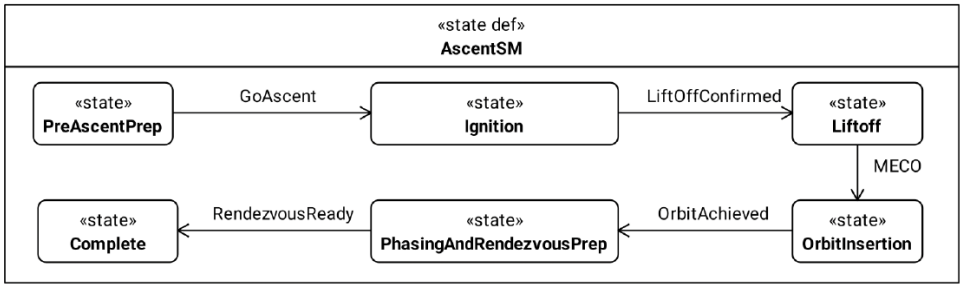
# Ascent Stage Subsystem Requirements

```
package 'Lunar Lander V2 Model' {
    package LunarLanderSubsystemsDecomposition {
        package Ascent_Stage {
            package Requirements {
                requirement def ASC_REQ_001_Capability {
                    doc /* The Ascent subsystem shall provide the capability to
                         * lift off from the lunar surface, insert into lunar orbit,
                         * and deliver the vehicle to a rendezvous-ready state. */
                }
                requirement def ASC_REQ_002_EngineControl {
                    doc /* The Ascent subsystem shall command ascent engine
                         * ignition, throttling or profiling as applicable,
                         * and shutdown per guidance. */
                }
                requirement def ASC_REQ_003_AttitudeAndRCS {
                    doc /* The Ascent subsystem shall control vehicle attitude in
                         * all ascent phases and provide translation impulses
                         * required for proximity operations. */
                }
                requirement def ASC_REQ_004_GuidanceNavigation {
                    doc /* The Ascent subsystem shall estimate state and compute
                         * guidance to meet target orbit conditions and rendezvous
                         * geometry within allocated errors. */
                }
                requirement def ASC_REQ_005_Interfaces {
                    doc /* The Ascent subsystem shall interface with power, thermal,
                         * communications, crew interface, and propellant/pressurant
                         * supplies required for ascent operations. */
                }
                requirement def ASC_REQ_006_FaultManagement {
```



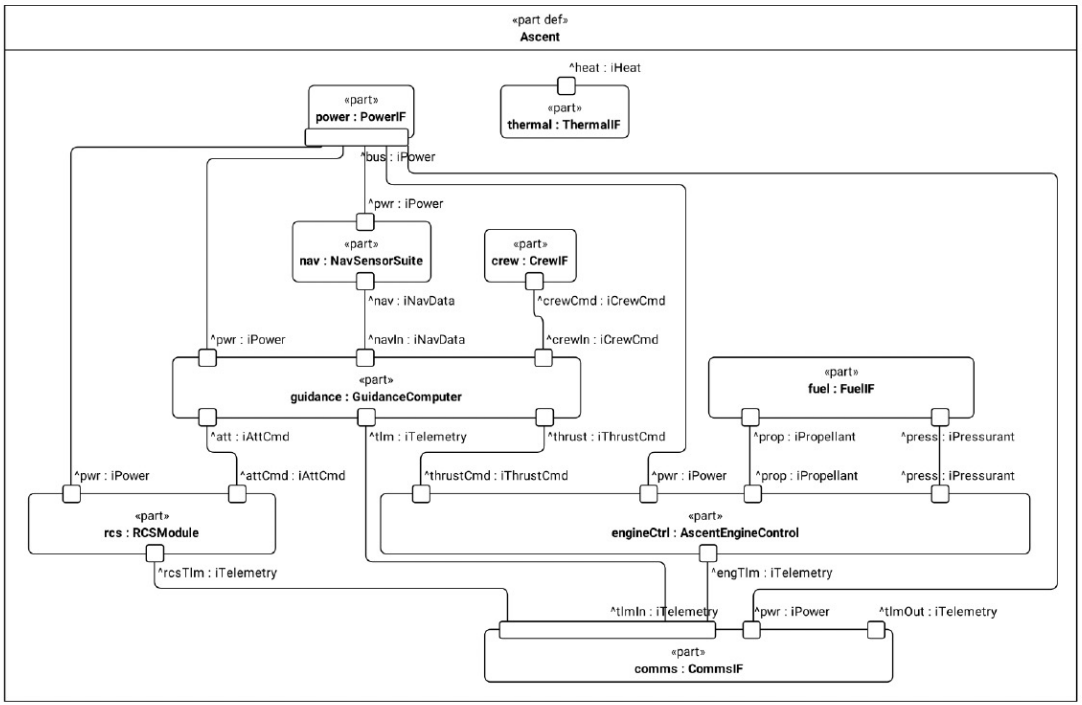
# Ascent Subsystem State Machine

```
package 'Lunar Lander V2 Model' {
    package LunarLanderSubsystemsDecomposition {
        package Ascent_Stage {
            package StateMachine {
                item def GoAscent;
                item def LiftOffConfirmed;
                item def MECO;
                item def OrbitAchieved;
                item def RendezvousReady;
                part def AscentSM_Owner {
                    exhibit state AscentSM;
                }
                state def AscentSM {
                    state PreAscentPrep;
                    state Ignition;
                    state Liftoff;
                    state OrbitInsertion;
                    state PhasingAndRendezvousPrep;
                    state Complete;
                    transition t0 first entryAction then PreAscentPrep;
                    transition t1 first PreAscentPrep accept GoAscent then Ignition;
                    transition t2 first Ignition accept LiftOffConfirmed then Liftoff;
                    transition t3 first Liftoff accept MECO then OrbitInsertion;
                    transition t4 first OrbitInsertion accept OrbitAchieved then
                    ↪ PhasingAndRendezvousPrep;
                    transition t5 first PhasingAndRendezvousPrep accept RendezvousReady
                    ↪ then Complete;
                }
```



# Ascent Subsystem Interfaces (IBD)

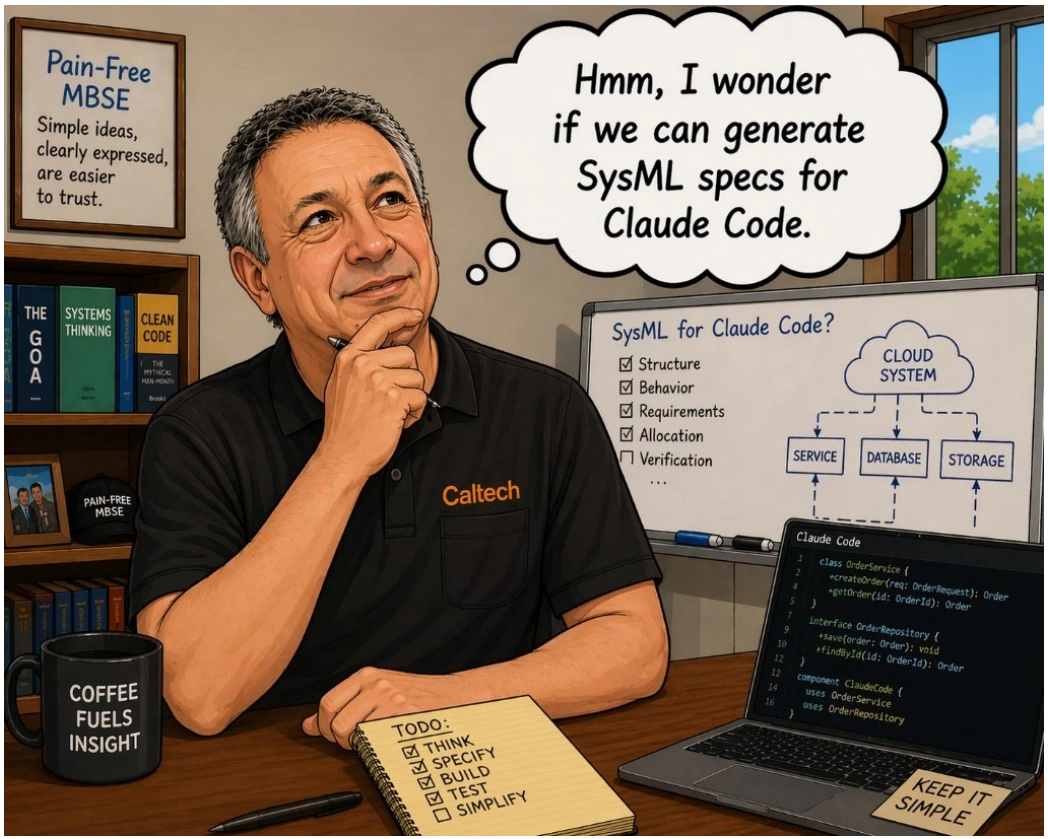
```
package 'Lunar Lander V2 Model' {
    package LunarLanderSubsystemsDecomposition {
        package Ascent_Stage {
            package Connections {
                private import ScalarValues::**;
                private import SI::**;
                interface def iPower;
                interface def iPropellant;
                interface def iPressurant;
                interface def iHeat;
                interface def iNavData;
                interface def iThrustCmd;
                interface def iAttCmd;
                interface def iCrewCmd;
                interface def iTelemetry;
                part def PowerIF {
                    port bus : iPower;
                }
                part def FuelIF {
                    port prop : iPropellant;
                    port press : iPressurant;
                }
                part def ThermalIF {
                    port heat : iHeat;
                }
                part def CrewIF {
                    port crewCmd : iCrewCmd;
                }
                part def CommsIF {
                    port pwr : iPower;
```



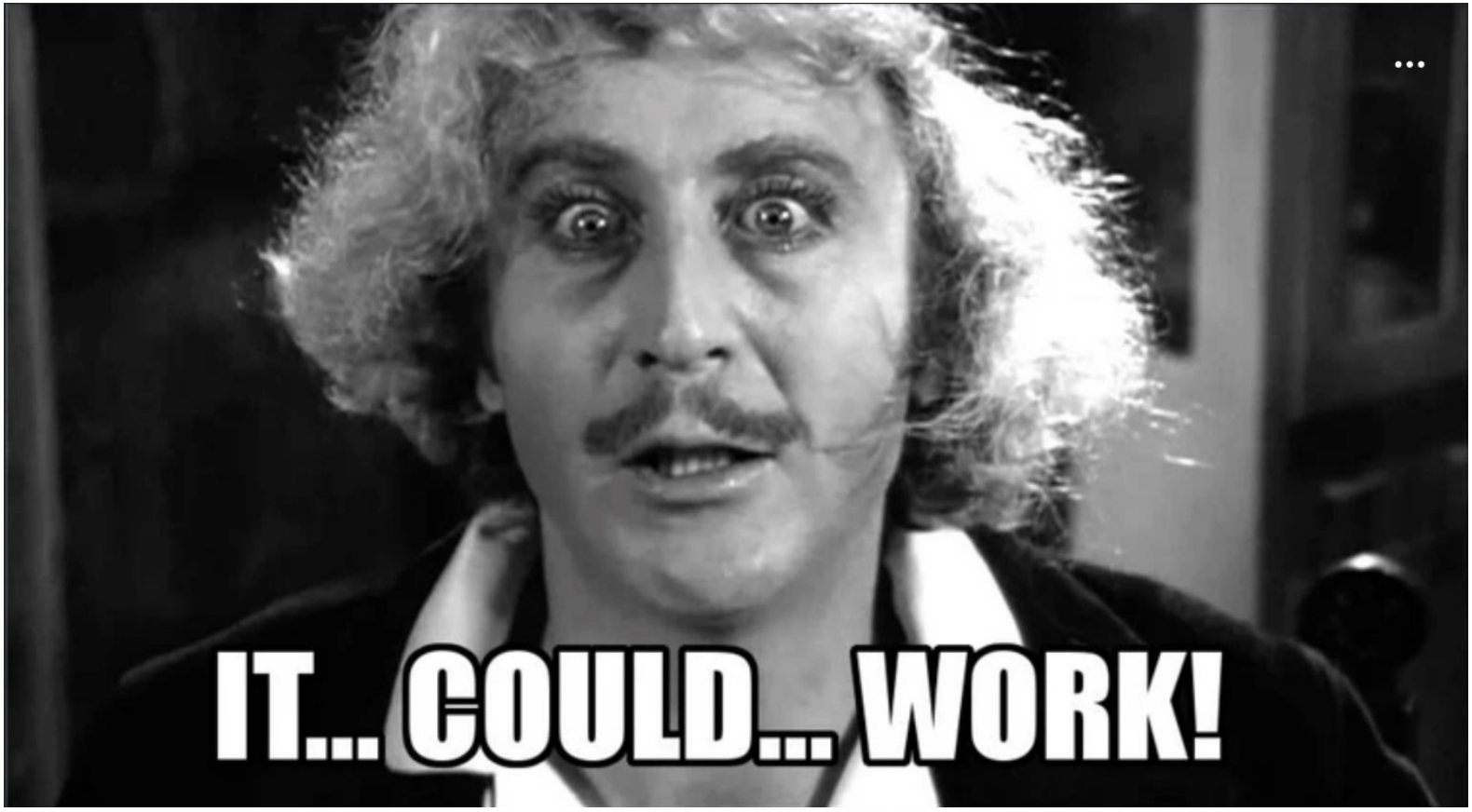
## Summing up

- Code generation templates provide a syntax reference that allows AI to **reliably** generate Cameo compliant SysML v2 code
  - This fully enables “AI as a Subject Matter Expert” that we described in AI Assisted MBSE
- We started with one template per SysML 1 diagram type
- As our reliability rates improved, we started combining templates
  - System Spec Template
  - Logical Architecture template (one subsystem at a time)
  - Build your own templates as needed
- We can now generate specs (parts, states, interfaces, requirements) a subsystem at a time
  - As always with AI, humans must review the results
- This represents an enormous productivity boost for systems engineers

# But wait…there's more…because a few weeks ago…



# Can Claude build an app from a SysML v2 spec???



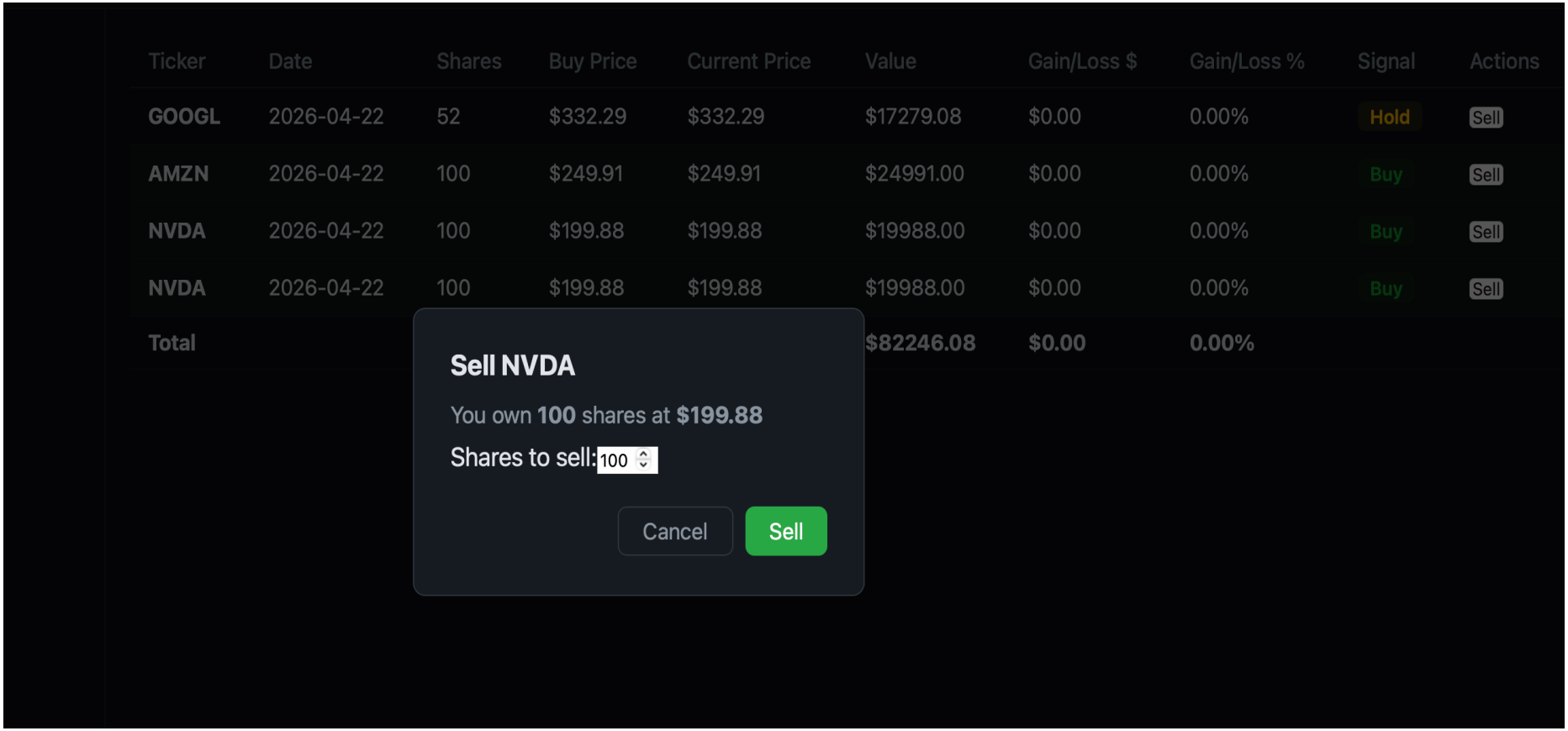
# We'll never know unless we try…

```
## 1. System Overview

**OVR_001** – StockPicker is a browser-based decision-support system for analyzing stock price data and
presenting advisory buy, hold, or exit signals.

The system is intended to help a user inspect:
- Price behavior
- Trend behavior
- Trend-change behavior
- Simulated portfolio performance using simple, explainable rules

### Build 1 Scope

Build 1 focuses on:
- Long-only strategies
- Simple Moving Average (SMA) smoothing
- Visualization of price and analysis curves
- Simulated portfolio support
- Explicit UI behavior
- Explicit testing structure
```

# Pull stock data from online sources

# Trend analysis

# Simulated portfolio to track investment strategy



# Hang on a tic…what's in that spec?

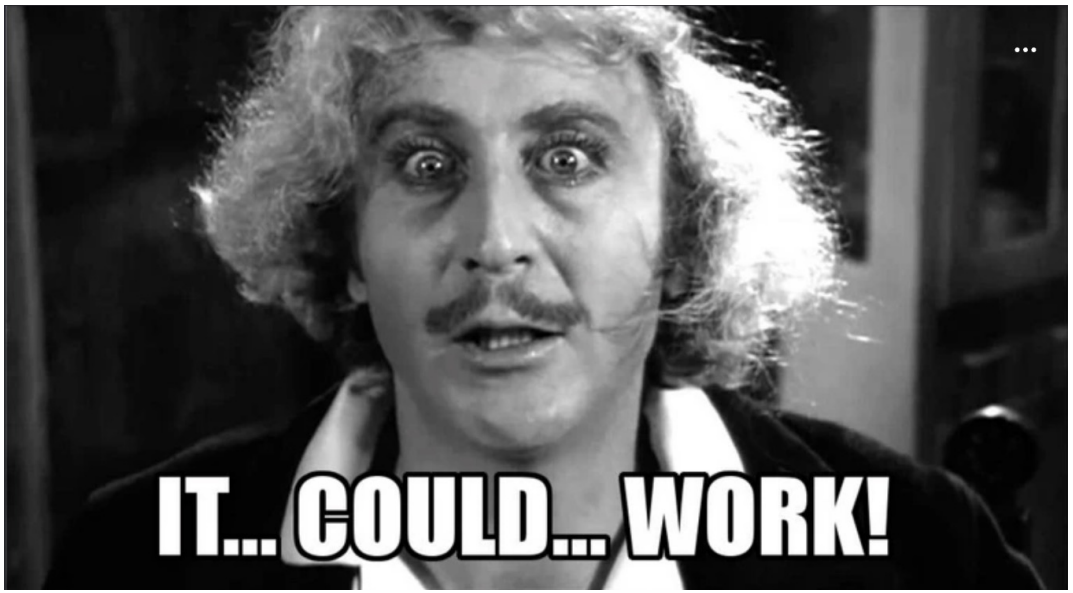
- Requirements
- Use Cases
- Domain Model (Glossary)
- State Machines
- UI Specs
- Data Models
- Testing Strategy

```
## Table of Contents

1. [System Overview](#1-system-overview)
2. [Glossary](#2-glossary)
3. [Functional Requirements](#3-functional-requirements)
   - 3.1 Core Data and Analysis
   - 3.2 User Interface
   - 3.3 Portfolio Simulation
   - 3.4 Algorithm Transparency
   - 3.5 Portfolio Display
   - 3.6 Deferred to Build 2
4. [Build Constraints](#4-build-constraints)
5. [Use Cases](#5-use-cases)
6. [System State Behavior](#6-system-state-behavior)
   - 6.1 Daily Cycle State Machine
   - 6.2 UI State Machine
   - 6.3 Data Lifecycle State Machine
7. [Algorithm Specification](#7-algorithm-specification)
8. [Data Model](#8-data-model)
9. [UI/UX Specification](#9-uiux-specification)
10. [Testing Specification](#10-testing-specification)
    - 10.1 Assertion Tests
    - 10.2 Thread Tests
    - 10.3 Coverage Matrix
```

# One prompt to generate the spec and three prompts to build the app?

3 prompts to Claude:

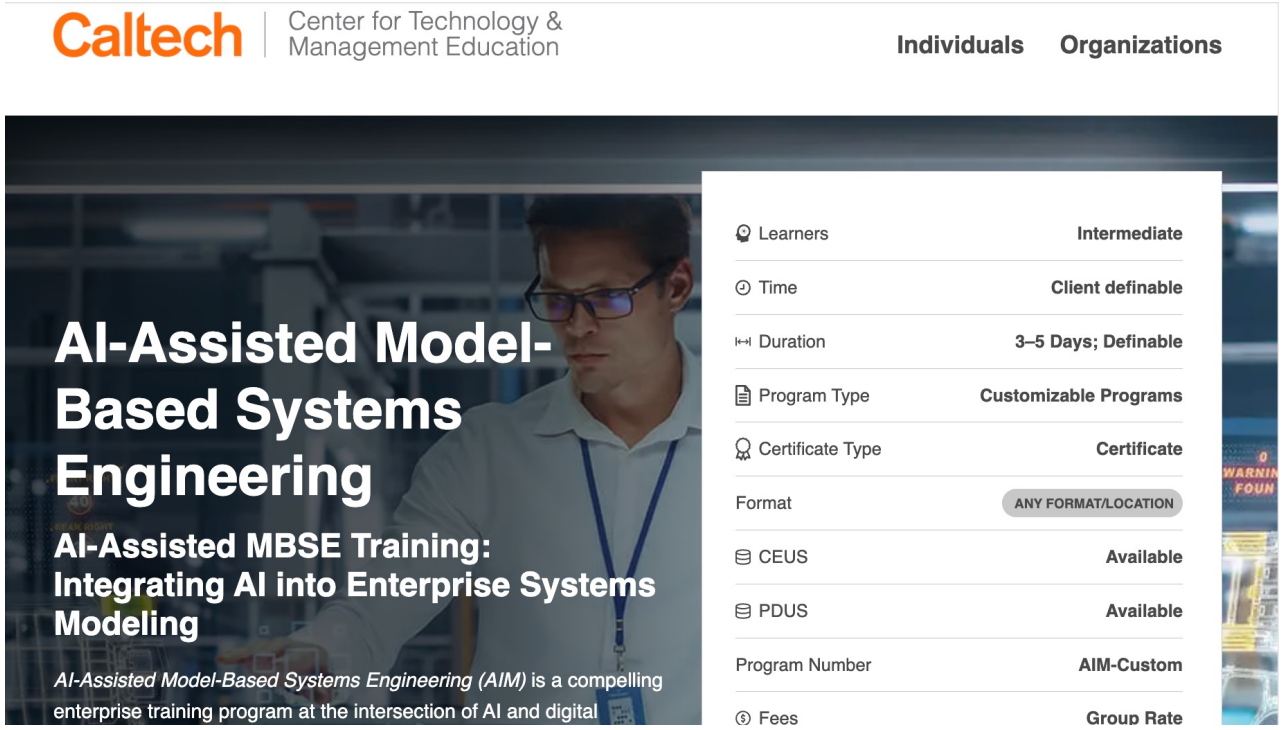
1) Build your markdown spec from this SysML v2 spec
2) Make a plan
3) Build the app



# Learn more…

*Dr. Rick Hefner, Caltech CTME:*
*rhefner@caltech.edu*

*Doug Rosenberg:*
*doug@parallelagile.com*




*https://ctme.caltech.edu/ai-assisted-model-based-systems-engineering-aim-certificate-custom.html*