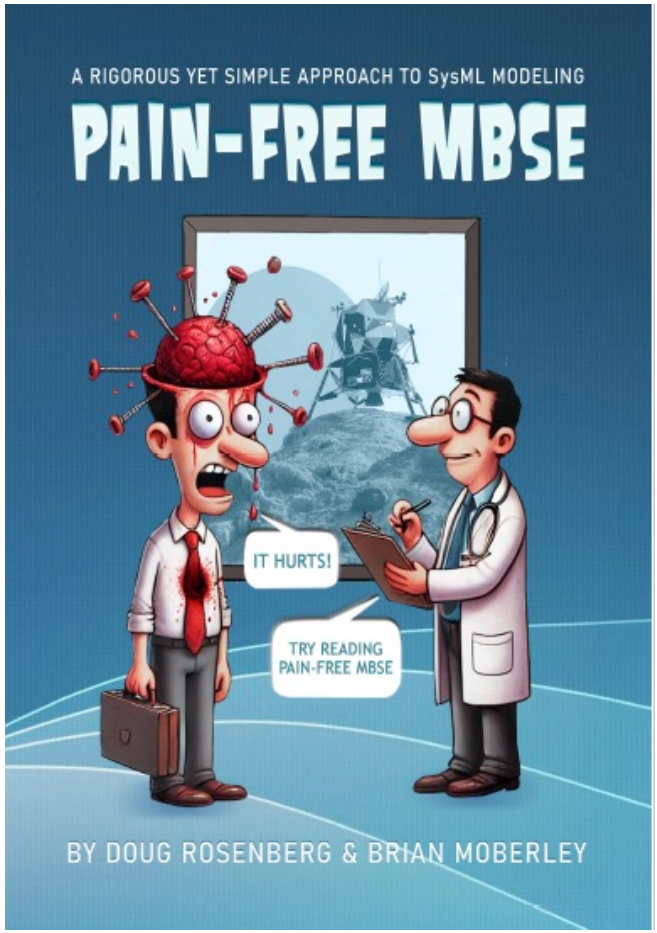
Caltech | Center for Technology & Management Education

Pain-Free MBSE

Rigor without the Mortis

Doug Rosenberg, Parallel Agile, Inc. and Caltech CTME

The pain level in SysML modeling is much too high.



Rigorous modeling does not require Rigor Mortis

The Pain-Free Manifesto

By Doug Rosenberg and Brian Moberley

As veterans of the MBSE community, we've spent decades immersed in the world of SysML modeling, working on some of the most complex engineering challenges across industries. Along the way, we've seen a troubling trend: too many "high-pain" approaches that complicate the modeling process, frustrate practitioners, and hinder the adoption of Model-Based Systems Engineering.

We declare today that SysML modeling does not need to be painful.

We believe that :

Thoroughness and completeness can coexist with simplicity. A well-structured model does not require unnecessary layers of complexity.
The purpose of MBSE is to solve problems, not create them. SysML should empower engineers to focus on design and analysis, not struggle with the tools and techniques.
Pain is not a requirement for precision. Clear methodologies and pragmatic practices can achieve rigor without sacrificing usability.
Rigor in modeling does not need to be accompanied by rigor mortis. Precision and clarity can be achieved without stifling progress or creativity.

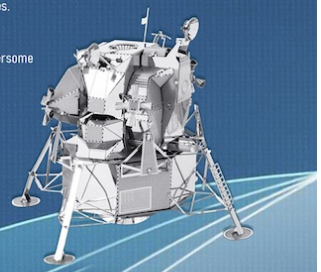
Our Commitment :

We vow to promote low-pain approaches to MBSE by :

Identifying the specific practices that cause unnecessary pain and offering actionable alternatives. Demonstrating how SysML can be applied effectively without sacrificing thoroughness or completeness.
Sharing practical, real-world examples like our lunar lander case study to show how MBSE can deliver value without the headaches.

Our Vision :

We envision a world where MBSE is not viewed as a cumbersome obligation but as a powerful, approachable methodology that engineers embrace with confidence and enthusiasm. Together, we can move beyond "high-pain" practices and unlock the true potential of SysML modeling.



Caltech

Evaluate Value to Pain Ratio (VPR) for all practices

Chapter 1 - 10% of the Practices Cause 90% of the Pain

The Value-to-Pain Ratio (VPR): A Metric for Evaluating MBSE Practices

Every systems engineering practice, technique, or modeling step comes with both value and pain—the key is determining whether the value outweighs the pain. To help MBSE practitioners make informed decisions about which practices to adopt and which to avoid, we introduce the Value-to-Pain Ratio (VPR) (Figure 1.1).

You have to evaluate Value but we can tell you about Pain

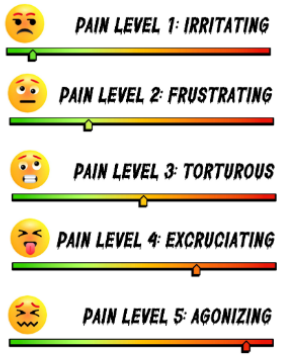
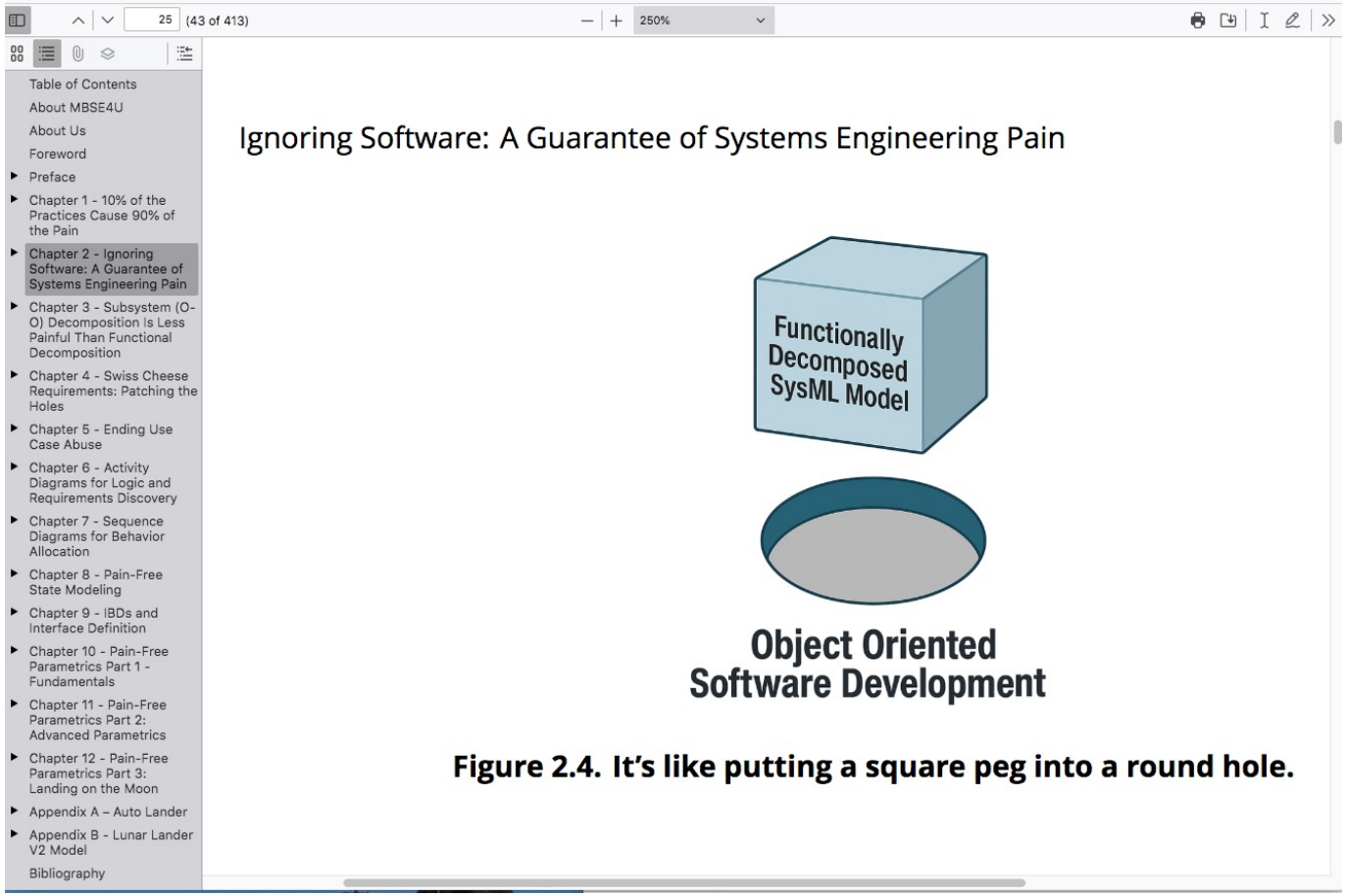


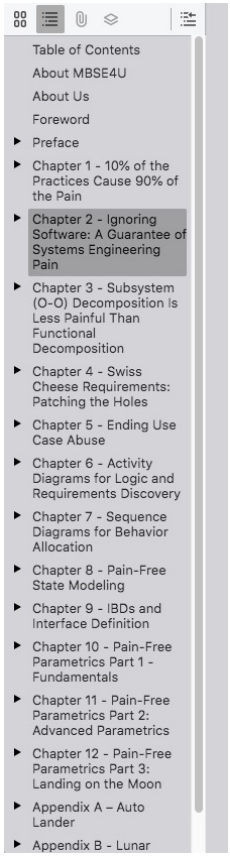
Figure 1.2. Five-Point Pain Scale

Level	Name	Description
1	*Irritating*	Mildly annoying but manageable.
2	*Frustrating*	Creates noticeable inefficiencies.
3	*Torturous*	Significantly complicates modeling efforts.
4	*Excruciating*	A major time sink with little payoff.
5	*Agonizing*	So painful it can derail an entire effort.

Ignoring software guarantees high pain



Functional Decomposition (if needed) informs Architecture



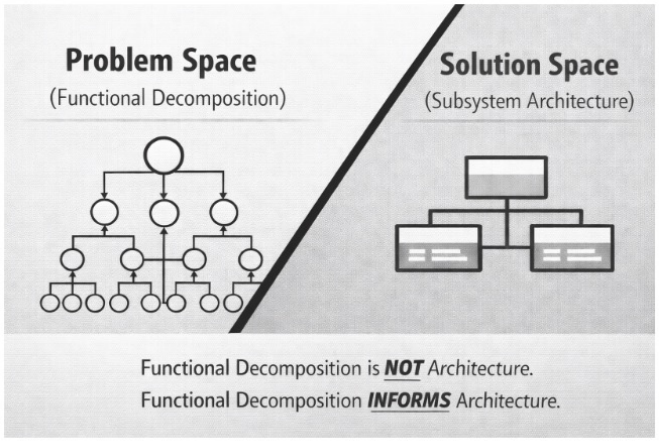


Figure 3.5. The use of the term functional architecture causes a lot of confusion

Unfortunately, systems engineering inherited two overloaded words—function and architecture—and combined them into a doubly overloaded phrase. This confusion is especially natural because in virtually every other domain—software, buildings, electronics, enterprises—the word architecture is used exclusively in a solution-space context. For 99% of the known universe, architecture means the arrangement of components, the structure of the solution, and the design decisions that shape implementation. So when systems engineers refer to a "functional architecture" in the problem space, it sounds like they are describing a

Architecture starts with Subsystems

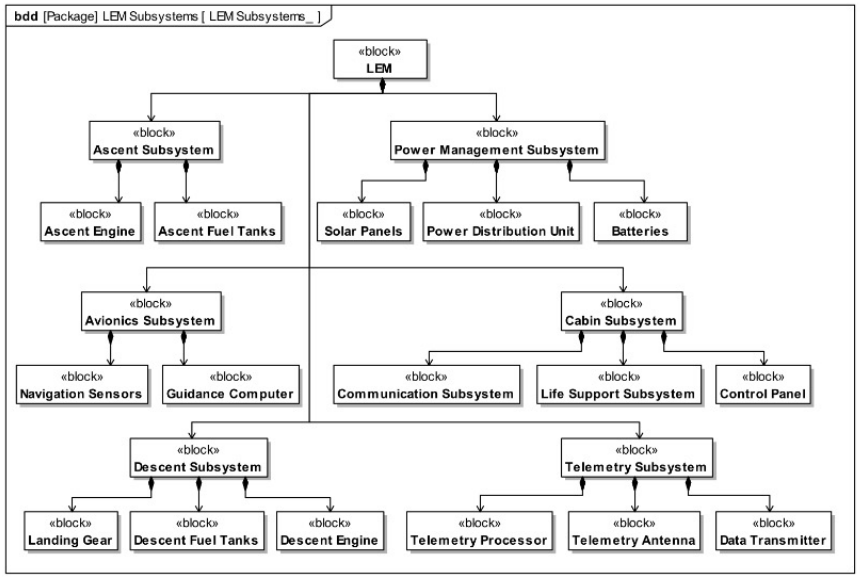
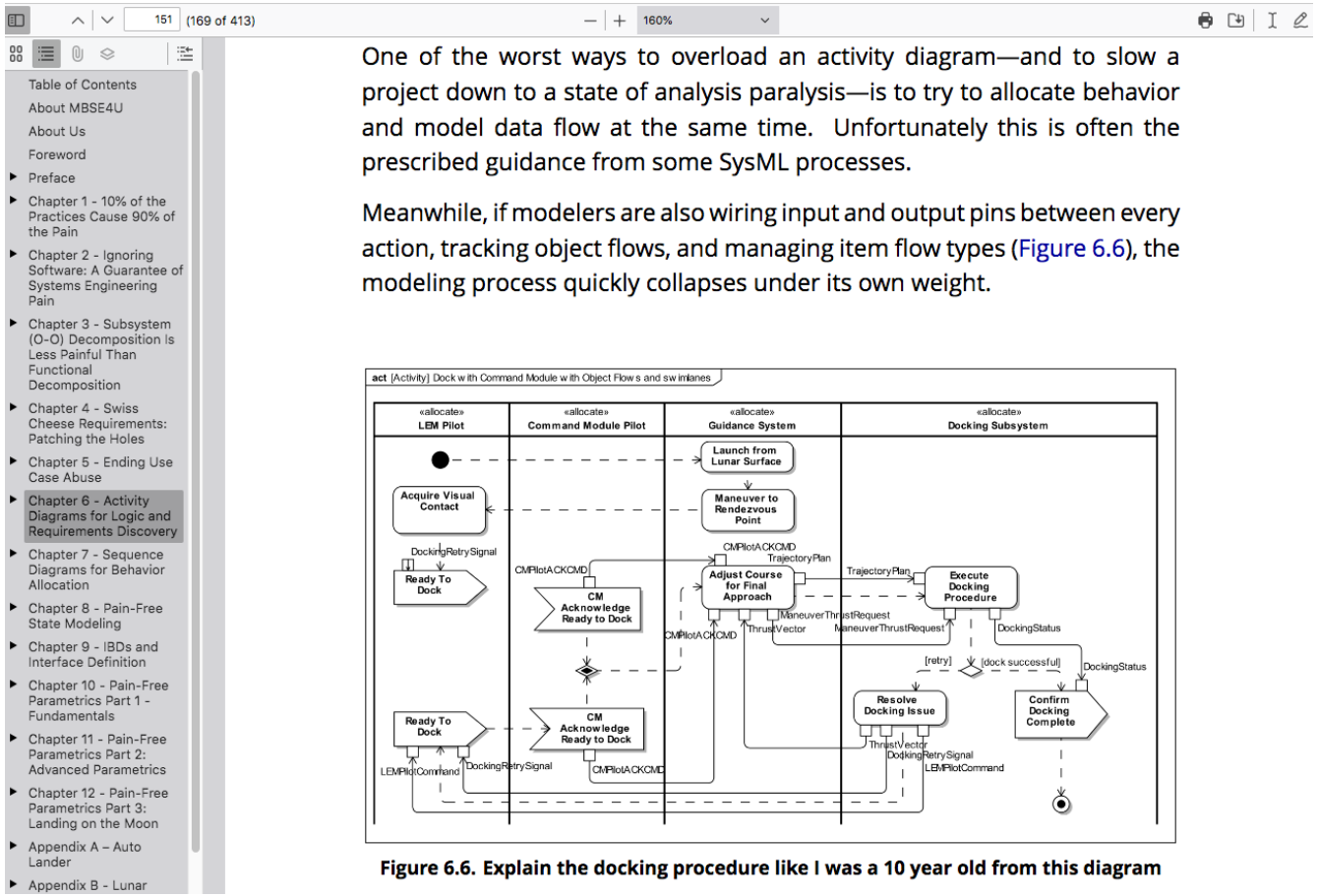


Figure 2.3. The first step after conceptual modeling is to introduce subsystems. Logical Architecture starts here.

Overloaded Activity Diagrams are painful



151 (169 of 413) 160%

Table of Contents
About MBSE4U
About Us
Foreword
Preface
Chapter 1 - 10% of the Practices Cause 90% of the Pain
Chapter 2 - Ignoring Software: A Guarantee of Systems Engineering Pain
Chapter 3 - Subsystem (O-O) Decomposition Is Less Painful Than Functional Decomposition
Chapter 4 - Swiss Cheese Requirements: Patching the Holes
Chapter 5 - Ending Use Case Abuse
Chapter 6 - Activity Diagrams for Logic and Requirements Discovery
Chapter 7 - Sequence Diagrams for Behavior Allocation
Chapter 8 - Pain-Free State Modeling
Chapter 9 - IBDs and Interface Definition
Chapter 10 - Pain-Free Parametrics Part 1 - Fundamentals
Chapter 11 - Pain-Free Parametrics Part 2: Advanced Parametrics
Chapter 12 - Pain-Free Parametrics Part 3: Landing on the Moon
Appendix A – Auto Lander
Appendix B - Lunar

One of the worst ways to overload an activity diagram—and to slow a project down to a state of analysis paralysis—is to try to allocate behavior and model data flow at the same time. Unfortunately this is often the prescribed guidance from some SysML processes.

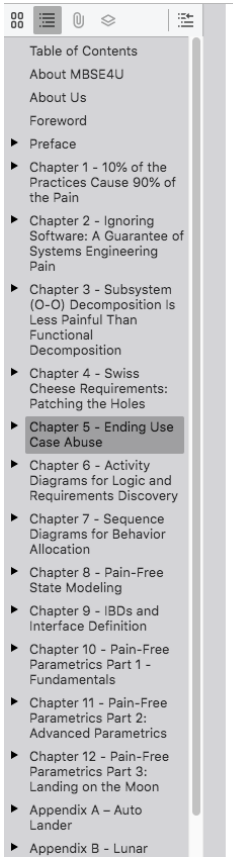
Meanwhile, if modelers are also wiring input and output pins between every action, tracking object flows, and managing item flow types (Figure 6.6), the modeling process quickly collapses under its own weight.

Figure 6.6. Explain the docking procedure like I was a 10 year old from this diagram

Is your Requirements Model full of holes?

Chapter 4 - Swiss Cheese Requirements: Patching the Holes

Use Cases help identify Requirements, not Subfunctions



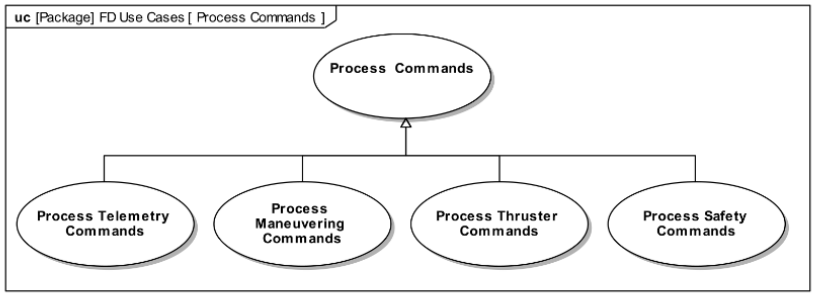
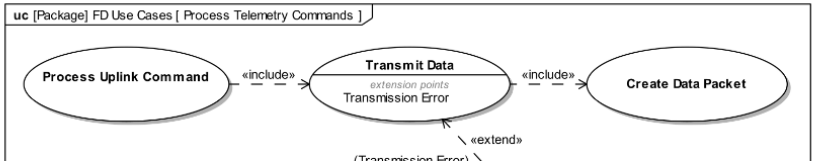
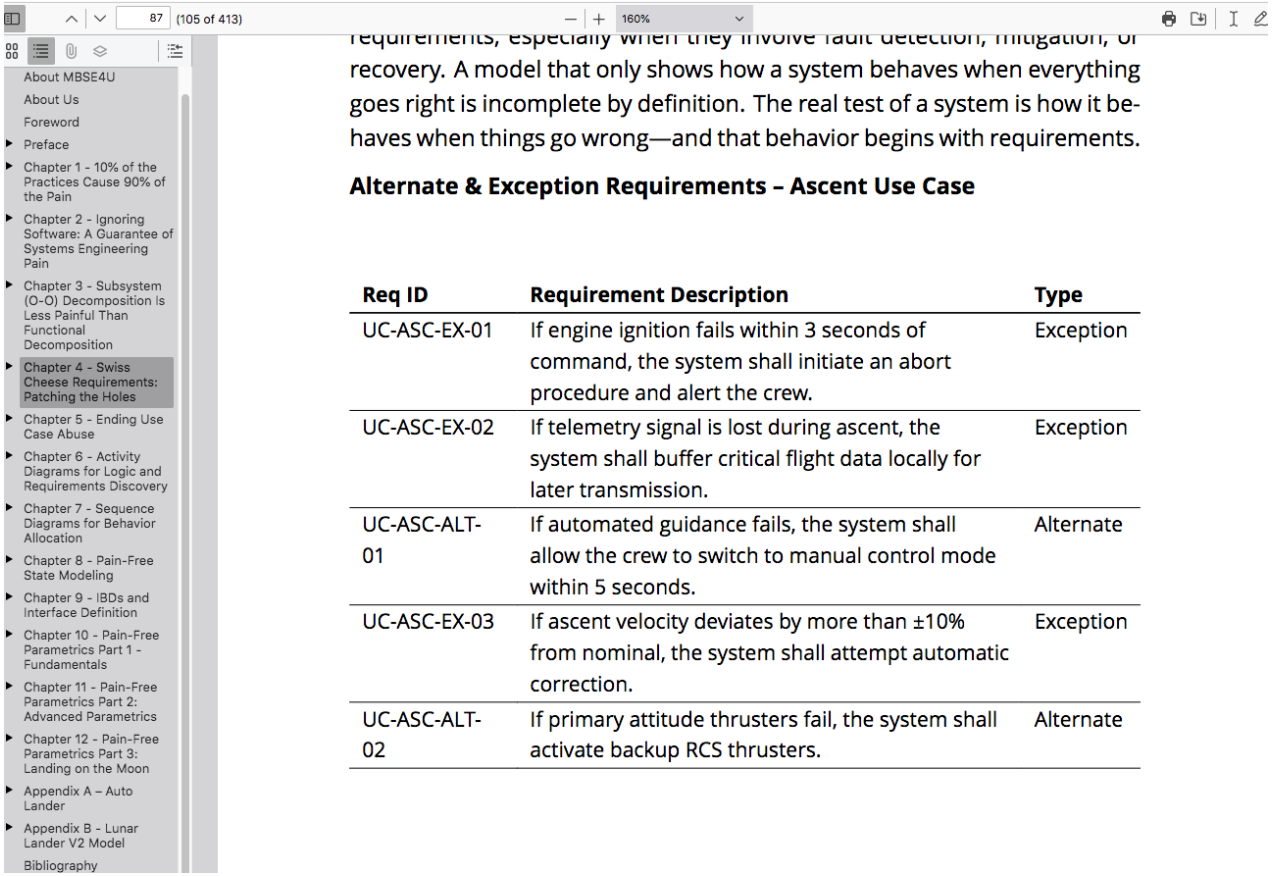


Figure 5.4. Just because you CAN do functional decomposition on use case diagrams doesn't mean you SHOULD

Often, each of these functionally decomposed steps is shown as a separate use case, with its own child diagram and fragments of behavior split across multiple narratives (Figure 5.5). The actor disappears. The story is lost. And instead of modeling what the user wants, the diagram becomes a map of what the system does internally. It's no longer a tool for understanding interactions—it's a mislabeled function chart.



If you don't write use case narratives, you miss Alternate and Exception Requirements



requirements, especially when they involve fault detection, mitigation, or recovery. A model that only shows how a system behaves when everything goes right is incomplete by definition. The real test of a system is how it behaves when things go wrong—and that behavior begins with requirements.

Alternate & Exception Requirements – Ascent Use Case

Req ID	Requirement Description	Type
UC-ASC-EX-01	If engine ignition fails within 3 seconds of command, the system shall initiate an abort procedure and alert the crew.	Exception
UC-ASC-EX-02	If telemetry signal is lost during ascent, the system shall buffer critical flight data locally for later transmission.	Exception
UC-ASC-ALT-01	If automated guidance fails, the system shall allow the crew to switch to manual control mode within 5 seconds.	Alternate
UC-ASC-EX-03	If ascent velocity deviates by more than ±10% from nominal, the system shall attempt automatic correction.	Exception
UC-ASC-ALT-02	If primary attitude thrusters fail, the system shall activate backup RCS thrusters.	Alternate

Sequence Diagrams help avoid overloaded Activity Diagrams

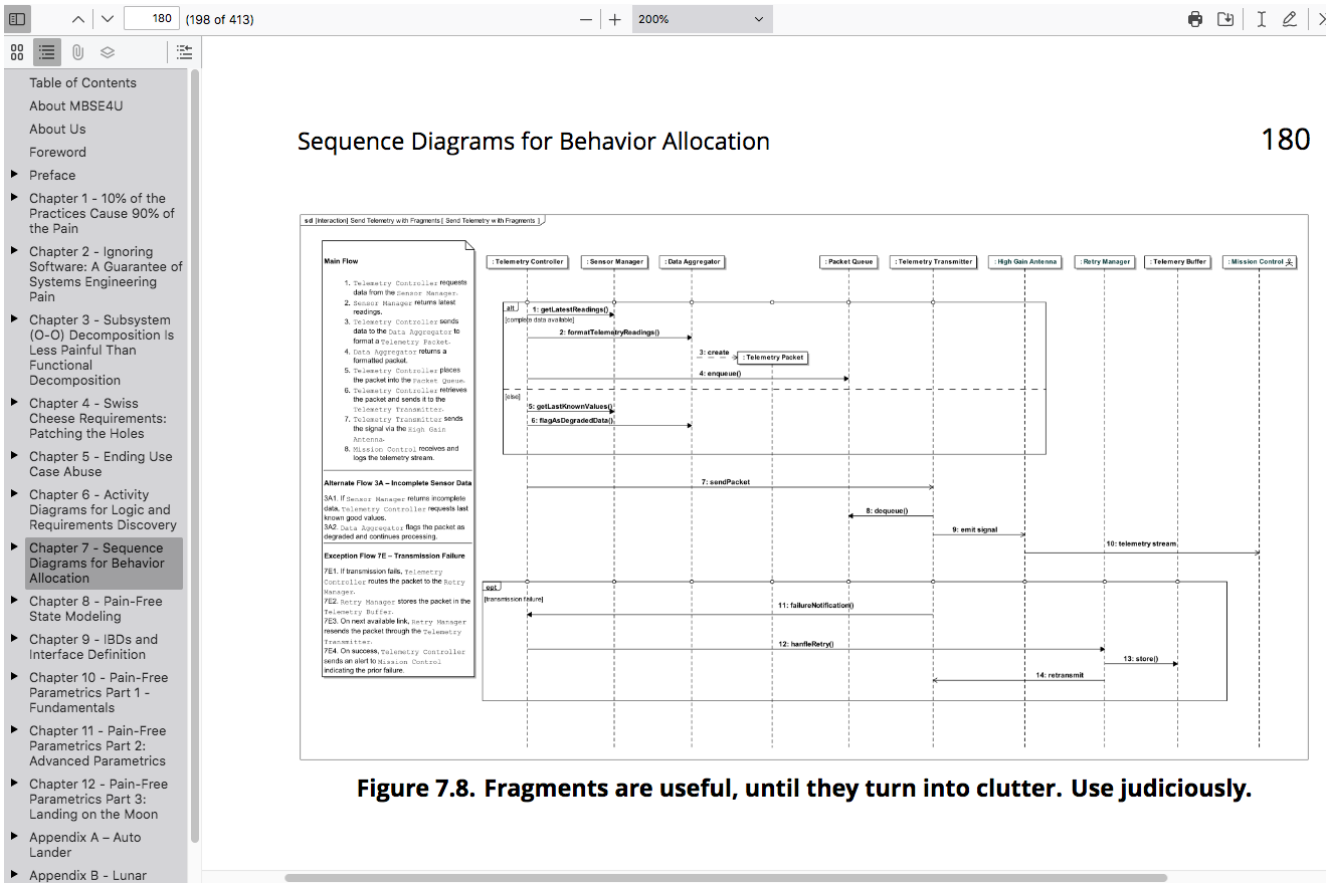
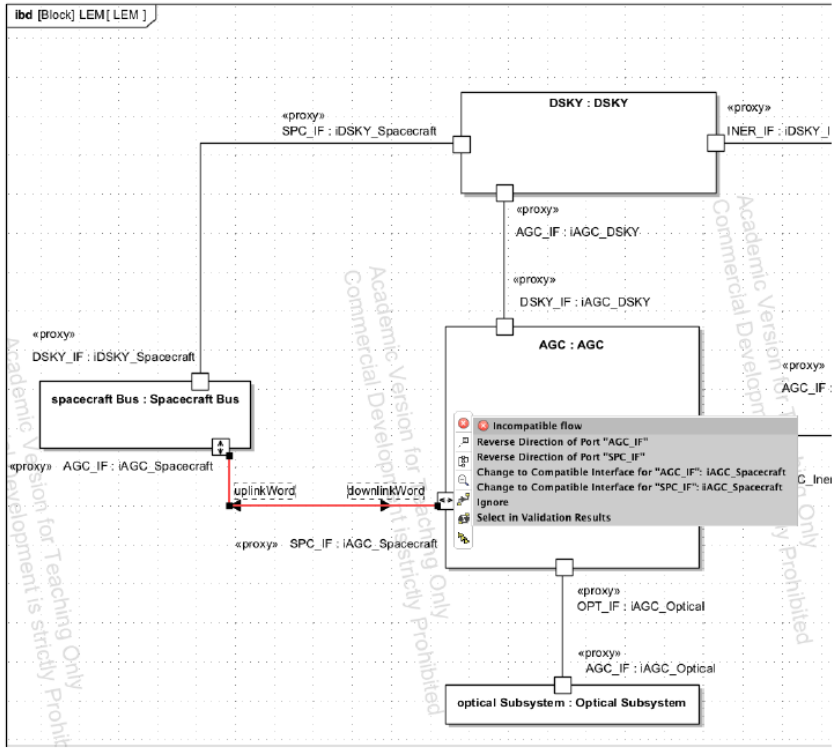


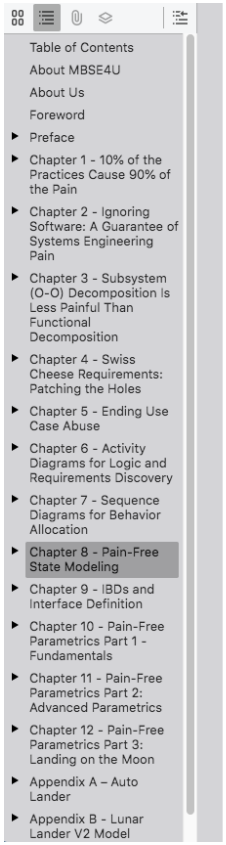
Figure 7.8. Fragments are useful, until they turn into clutter. Use judiciously.

Public enemy #1 – item flow violations on IBDs



The source of the problem was simple - we had neglected to conjugate the port on the Spacecraft Bus. Clicking on the error gives us a well-intended set of options on how to fix it. The typical response is to

Many people don’t realize the power of State Machines



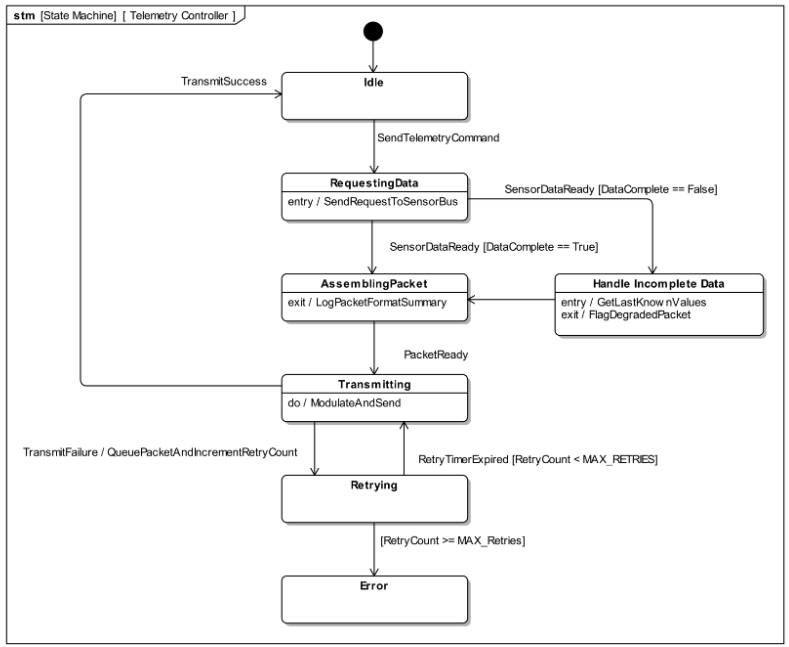
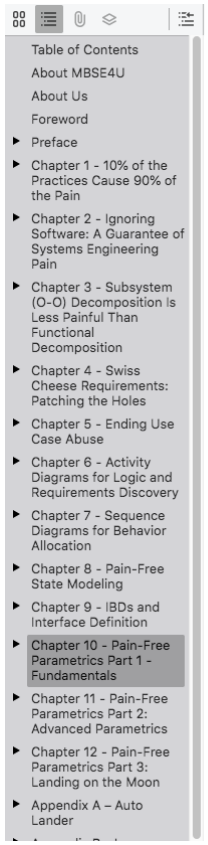


Figure 8.1. Telemetry State Machine with entry, exit, and do behavior, transitions, and guards

We've got 3 Chapters and an Appendix on Parametrics



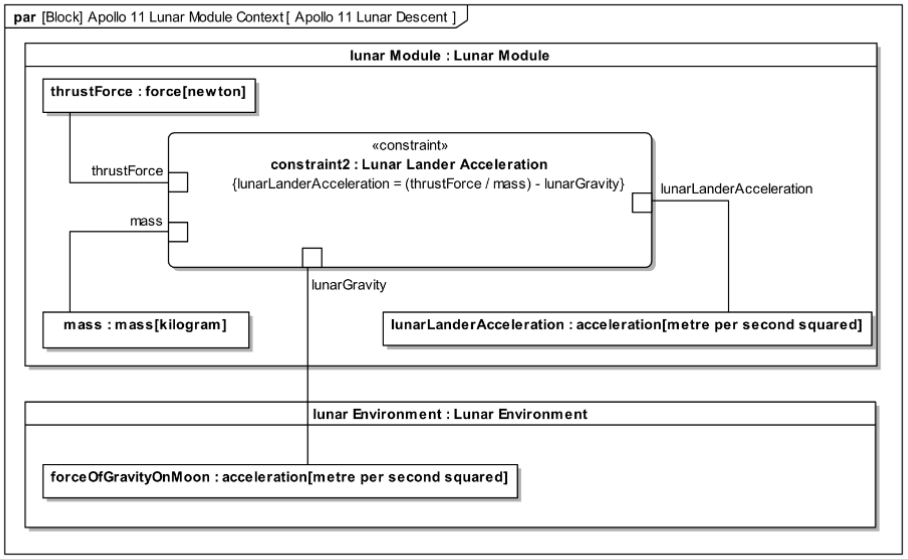


Figure 12.3. Parametric Diagram showing how acceleration is calculated

To visualize this behavior, create a time series chart to plot Altitude by right-clicking the value property and selecting "Show in Time Series Chart" (Figure 12.4).

Our “Lunar Lander Game” simulation is a free download

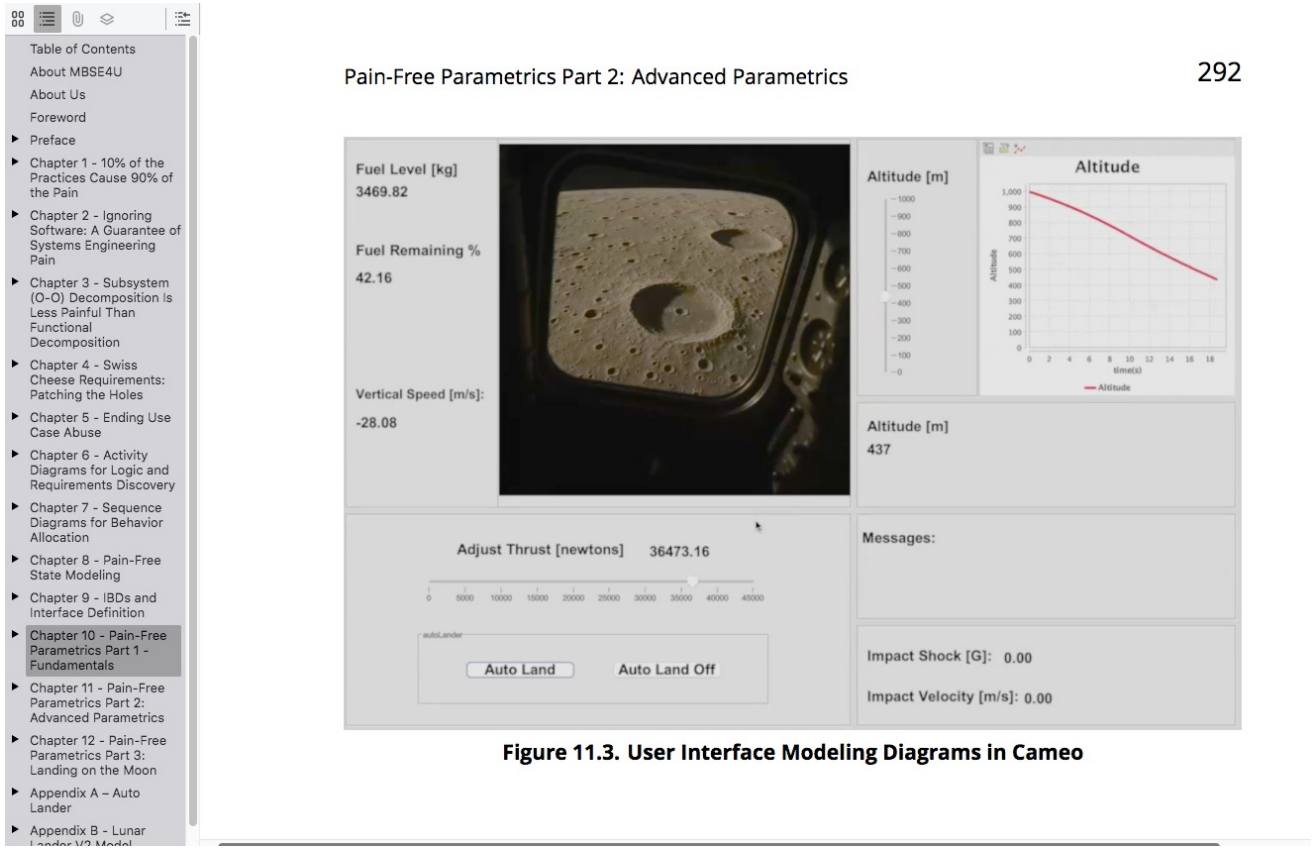
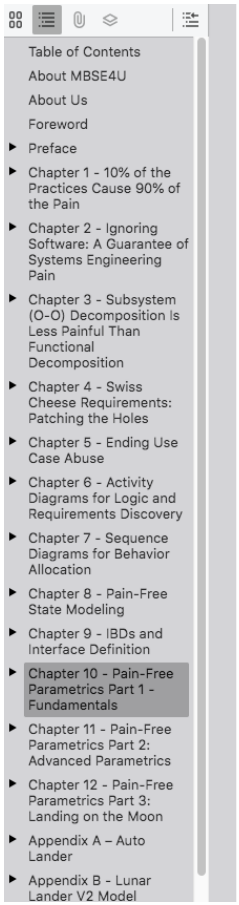


Figure 11.3. User Interface Modeling Diagrams in Cameo

Autolander function includes a P-I Control Loop



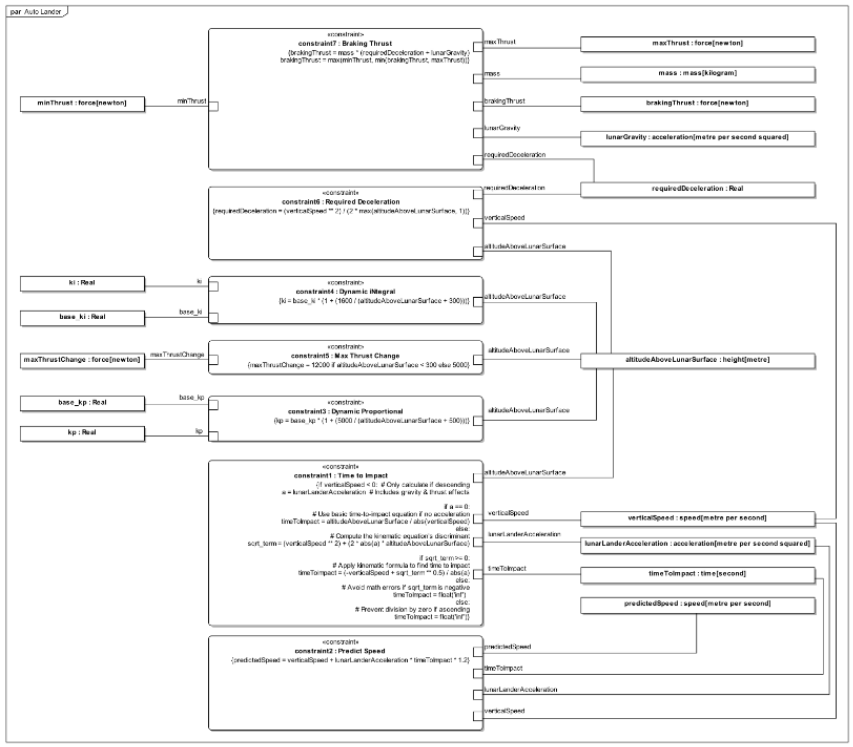
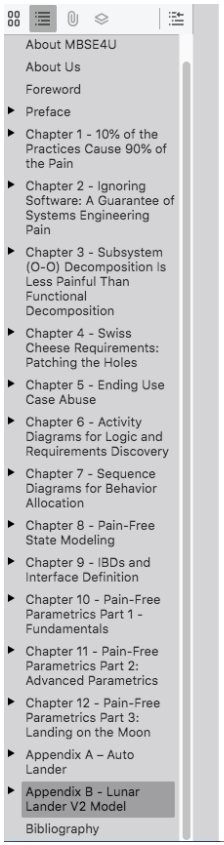


Figure 13. The seven Autolander constraints shown on a Parametric Diagram.

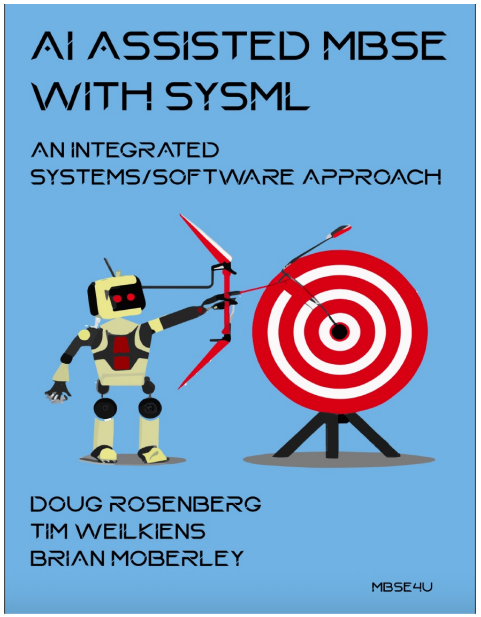
We introduce AI Assisted MBSE with Lunar Lander SysML v2



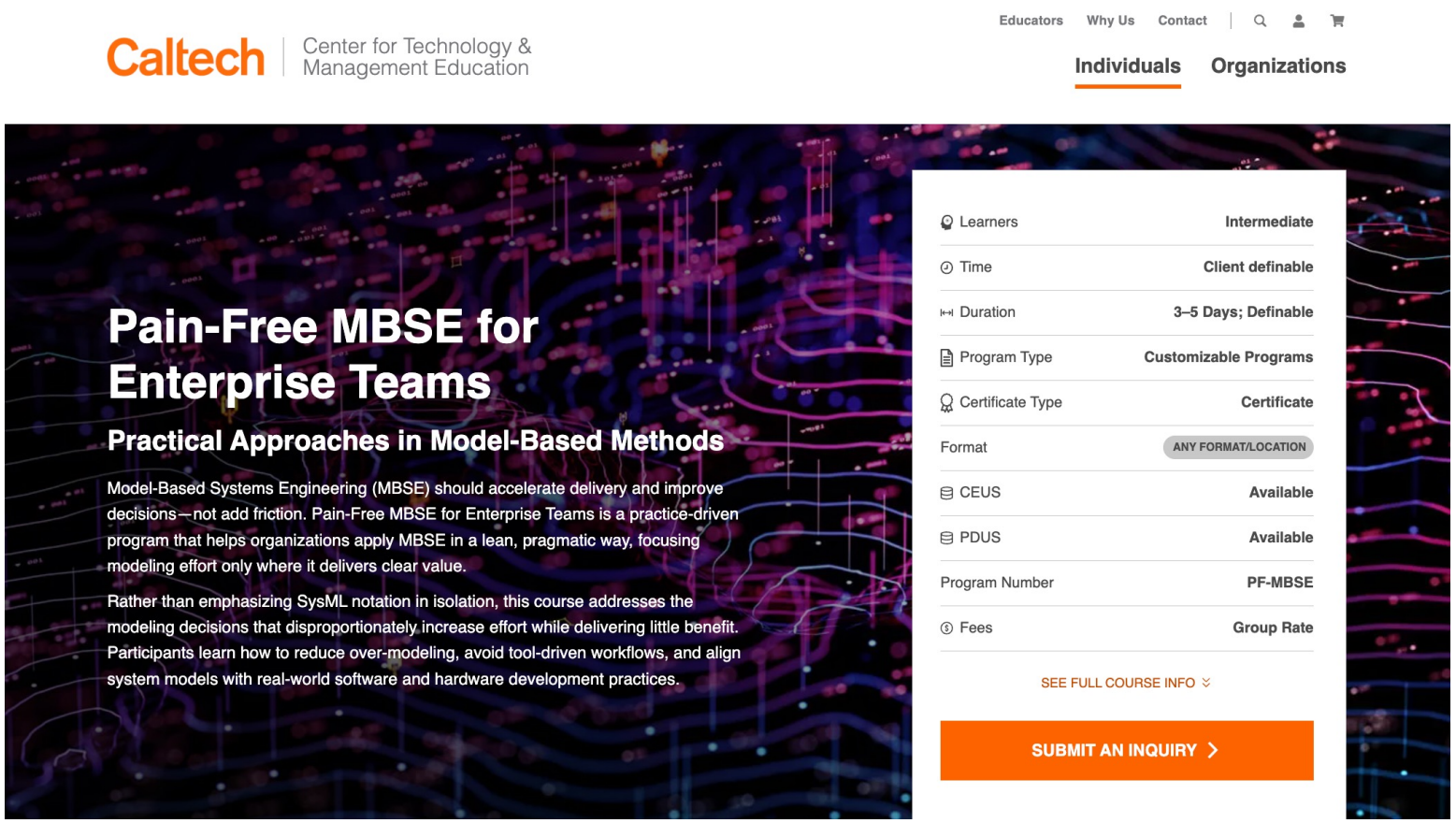
Domain Model

Figure 19. Lunar Lander Domain Model (Textual Notation)

```
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 {
```



Learn More: Design a Lunar Lander (without pain)



Caltech

https://ctme.caltech.edu