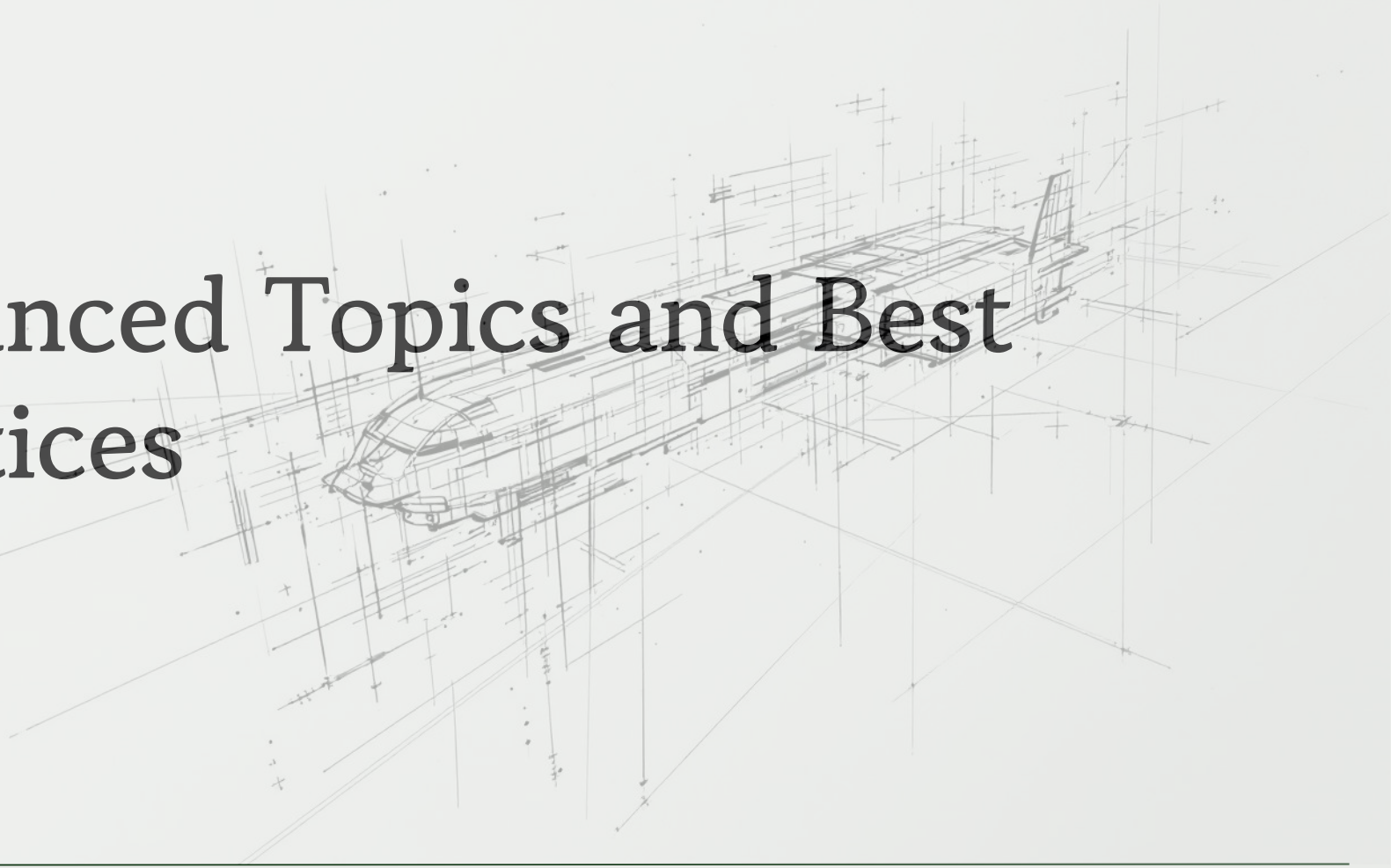
Advanced Topics and Best Practices

Week 14



Week Objective and Motivation

This week, we focus on how to organize, modularize, and reuse SysML v2 models.
Large-scale models must be structured to stay clear, maintainable, and scalable.
Key goals this week:

- Use packages, namespaces, and imports to structure complex models
- Create reusable libraries and definitions
- Apply profiles to extend SysML for your domain
- Learn instance specifications to capture concrete configurations
- Use FMEA and FTA as analysis techniques within the model
- Introduce Human-in-the-Loop, Software-in-the-Loop, and Hardware-in-the-Loop as part of validation

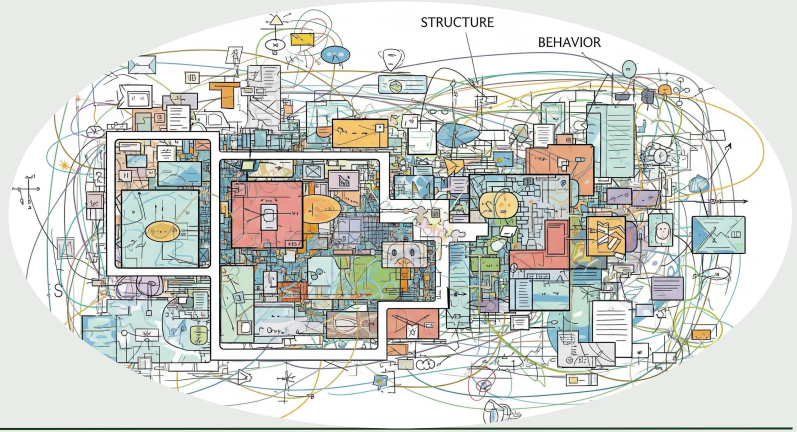
Modular modeling supports collaboration, reuse, and system evolution.

Problems with Monolithic Models

Unstructured models create major barriers as systems scale:

- Hard to navigate or search effectively
- Risk of naming collisions and inconsistent usage
- No clear boundary between requirements, structure, and behavior
- Difficult to assign ownership and collaborate across teams
- Weak support for reuse, analysis, and traceability

Without modularization, the model stops being a living engineering tool and turns into a liability.



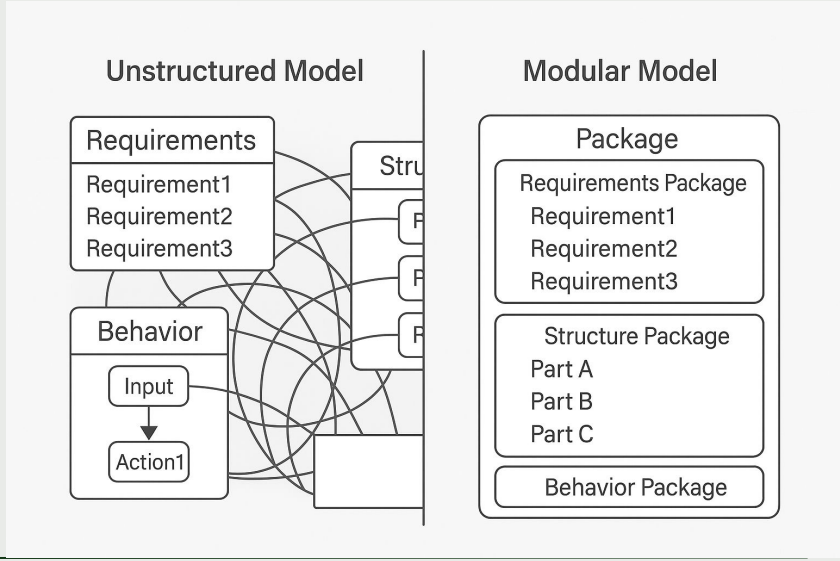
From Monolithic to Modular Models

Monolithic models:

- Flat structure with everything in one space
- Difficult to scale, reuse, or analyze
- Stakeholders overwhelmed by irrelevant details

Modular models:

- Organized into packages, namespaces, and libraries
- Clear ownership and separation of concerns
- Easier reuse, analysis, and validation
- Stakeholder-specific views without duplicating content

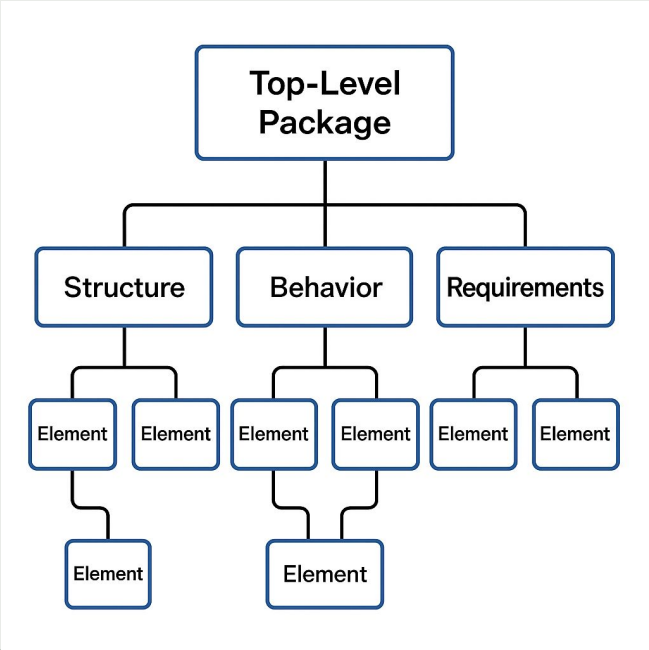


Modularization in SysML v2

SysML v2 provides strong constructs for structuring large models:

- **Packages**: group related elements into logical units
- **Namespaces**: prevent name collisions and clarify ownership
- **Imports**: bring in elements from libraries or other packages
- **Libraries**: define reusable parts, actions, attributes, and patterns

Modularization makes complex models navigable, reusable, and scalable.



Reusable Libraries and Profiles

Reusable assets are key to scaling SysML v2:

Libraries

Packaged definitions for parts, actions, attributes, ports, and units.

Profiles

Domain-specific extensions of SysML for various industries.

Consistency

Shared definitions keep models aligned across teams and projects.

Efficiency

Reuse reduces duplication and accelerates development processes.

When to Use Libraries vs. Profiles

Use a Library when…

- You need a common set of parts, actions, attributes, or units
- The content is reusable across projects without change
- The focus is on standardization and efficiency

Use a Profile when…

- You need to tailor SysML for a specific domain (e.g., aerospace, automotive)
- You want to add stereotypes, constraints, or domain rules
- The goal is to shape how SysML is applied, not just reuse content

Libraries and Profiles in Action

Example – UAV System Model

- **Library**: Provides reusable elements
  - Parts: Engine, Battery, Sensor, Wing
  - Attributes: Mass, Power, Range
  - Ports: FuelIn, DataLink
- **Profile**: Customizes SysML for UAV domain
  - Stereotypes: «FlightCritical», «Payload», «CertificationReq»
  - Rules: All «FlightCritical» parts must have redundancy defined

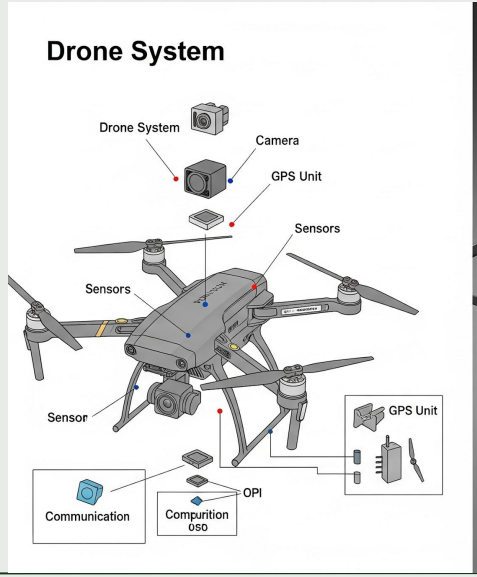
Together:

- The **library** supplies shared building blocks
- The **profile** enforces UAV-specific semantics and rules

Instance Specification – Concept Introduction

Definitions describe general types. Instances capture concrete examples.

- A **definition** says what something is in general (e.g., *Drone System*).
- An **instance specification** represents a specific occurrence (e.g., *Drone #001 with 2-hour battery*).
- Instances give models real-world context for configuration, simulation, and validation.



Instance Specification – Examples in Practice

From Definition → to Instance:

- **Definition**: *Drone System*
  - Mass: variable
  - Battery: 1–2 hr
  - Payload: optional sensor
- **Instance**: *Drone #001*
  - Mass: 7.2 kg
  - Battery: 2 hr
  - Payload: EO Camera

Other examples:

- **Definition**: Engine → **Instance**: Engine SN-1032
- **Definition**: Mission Profile → **Instance**: Recon Mission (Altitude 1000 m, 60 min)

Instance Specification vs. Part Usage

SysML v2

- **Part Usage** = a structural feature in a definition
  - Example: *drone1 : Drone System* inside a system definition
- **Instance Specification** = explicit instance of a definition with fixed values
  - Example: *Drone SN-001* with 7.2 kg mass, 2 hr battery

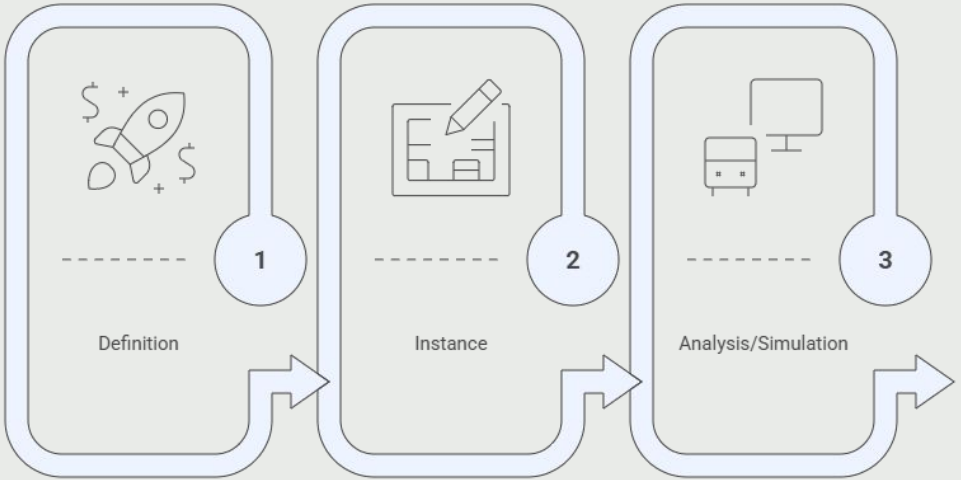
Key difference:

- **Part Usage** is about *structure in the model definition*
- **Instance Specification** is about *real, concrete occurrences*

Instance Specifications in Analysis and Simulation

Why instances matter in practice:

- **Configuration**: capture exact system builds (e.g., Drone SN-001 vs. Drone SN-002)
- **Trade Studies**: compare different configurations with fixed values
- **FMEA / FTA**: analyze failure modes and fault paths on specific instances
- **Simulation**: feed concrete values into Human-in-the-Loop, Software-in-the-Loop, and Hardware-in-the-Loop setups



Trade Study with Instance Specifications

Definition: Drone System

- Mass: variable
- Battery: 1–2 hr
- Payload: sensor optional

Instances for Trade Study:

- *Drone_A*: 7.2 kg, 2 hr battery, EO Camera
- *Drone_B*: 6.5 kg, 1.5 hr battery, IR Camera
- *Drone_C*: 8.0 kg, 2 hr battery, EO + IR Cameras

Use cases:

- Compare endurance vs. payload weight
- Evaluate reliability with FMEA
- Simulate missions with different sensor packages

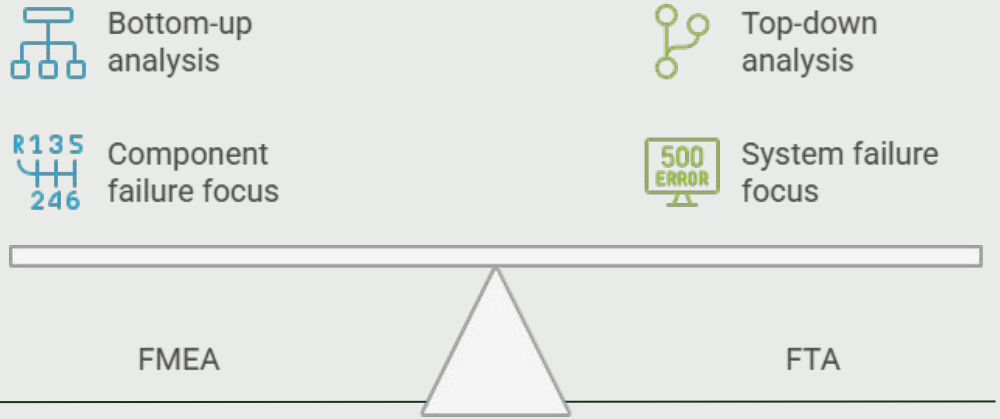
FMEA and FTA – Concept Introduction

Reliability analysis is essential for complex systems.

- **FMEA (Failure Mode and Effects Analysis)**: bottom-up, studies how each component can fail and its effect on the system.
- **FTA (Fault Tree Analysis)**: top-down, starts from a system failure and traces possible causes.

Both methods complement each other:

- FMEA helps identify vulnerabilities early.
- FTA helps assess risk pathways and probabilities.



Applying FMEA in SysML v2

Failure Mode and Effects Analysis (FMEA):

- Link **part definitions** and **instance specifications** to possible failure modes
- Record attributes: severity, occurrence, detection method
- Trace failures from component level to system effect
- Use libraries for common failure patterns (e.g., battery depletion, sensor dropout)

FMEA in SysML v2 keeps reliability analysis connected to the system model.

Applying FTA in SysML v2

Fault Tree Analysis (FTA):

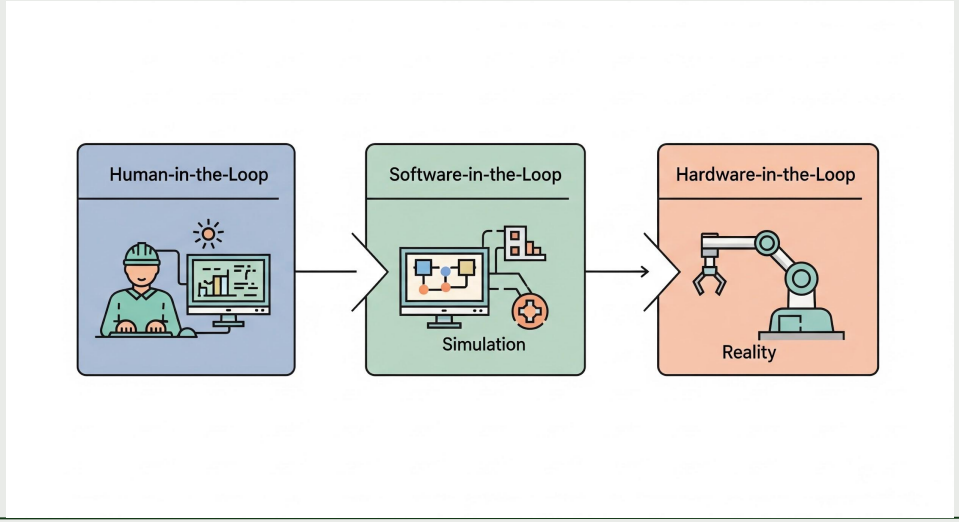
- Start with a **top-level system failure** (e.g., drone crash)
- Decompose into intermediate and basic fault events
- Use logical gates (AND, OR) to trace combinations of failures
- Connect fault events back to **parts, behaviors, or requirements** in the SysML model
- Enable probability or risk analysis within the model context

X-in-the-Loop – Concept Introduction

“X-in-the-Loop” = progressive testing and validation:

- **Human-in-the-Loop (HITL)**: operators interact with the model or simulation
- **Software-in-the-Loop (SITL)**: real code runs in the simulated environment
- **Hardware-in-the-Loop (HIL)**: physical components integrated into the loop

Goal: close the gap between abstract models and real-world performance.



Human-in-the-Loop & Software-in-the-Loop

Human-in-the-Loop (HITL):

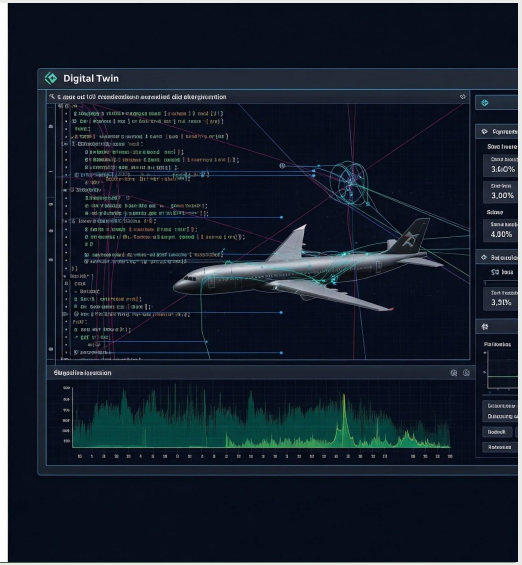
- Operators interact with simulated system behavior
- Validates usability, workload, and decision-making
- Helps align system design with human factors

Software-in-the-Loop (SITL):

- Real control software runs inside the simulation
- Tests logic and algorithms without full hardware
- Enables rapid iteration and early bug detection

Hardware-in-the-Loop (HIL)

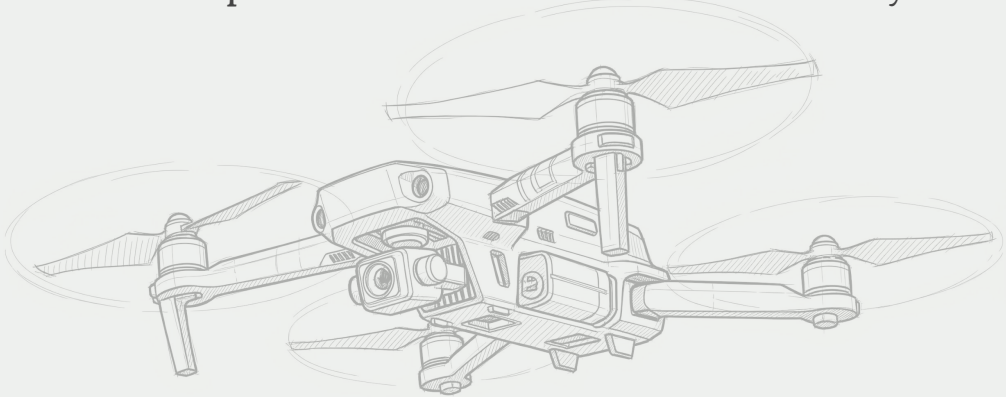
- Integrates real hardware with the simulated system
- Validates interfaces, timing, and performance under realistic conditions
- Finds issues before full system assembly
- Bridges the final step from digital model to physical prototype



Summary of Week 14

This week we advanced from tidy models to professional-scale MBSE practice:

- Organized large models using **packages, namespaces, and imports**
- Built **reusable libraries** and applied **profiles** for domain-specific modeling
- Created **instance specifications** to capture real configurations and enable trade studies
- Applied **FMEA and FTA** for reliability and safety analysis inside SysML v2
- Introduced **Human-, Software-, and Hardware-in-the-Loop** methods to connect models with reality



QUESTION!