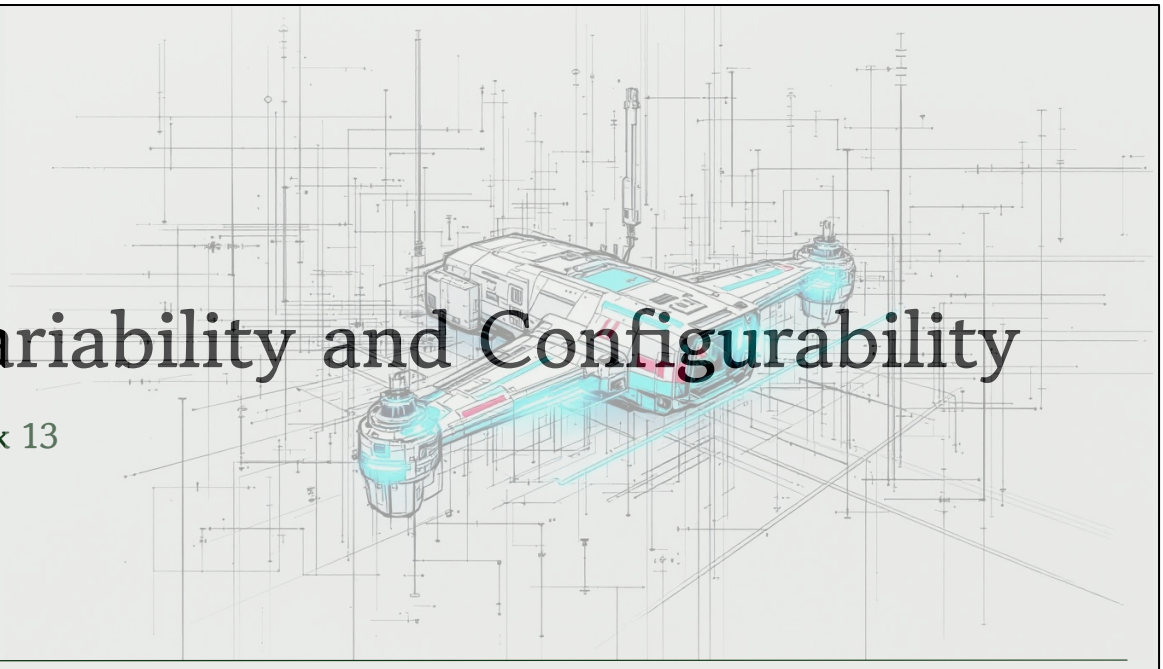
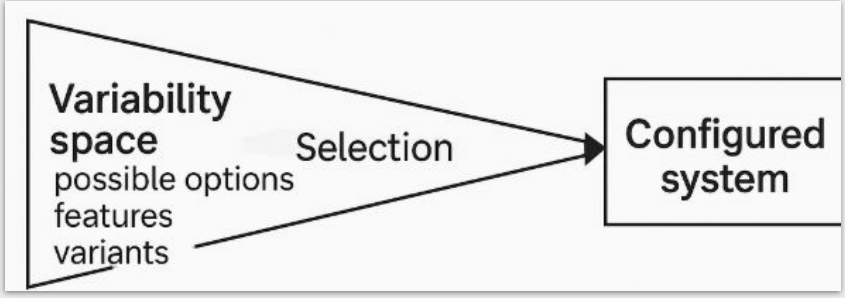


Variability and Configurability

Week 13

Variability and Configurability

This week, we explore how to model **variability** and **product configurations** within a single system model. You'll learn how to use variation and variant elements in SysML v2 to support flexible architectures and multiple drone missions.
By the end of this session, you'll be able to define variation points and build configurable structures and behaviors that adapt to different needs.



What Is Variability in Systems?
Welcome to Week 13 — where we shift from modeling a **fixed system** to designing a **flexible system family**.
In real-world engineering, we often don't build just one version of a system. We build **product lines** — multiple variants of a system that share a common foundation but differ in specific features. Think of:

- A drone with or without a camera
- One built for delivery, another for surveillance
- Or even two versions that differ only in flight range or autonomy

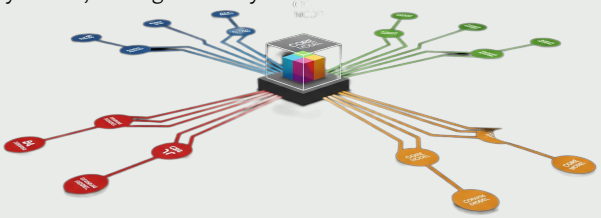
SysML v2 gives us tools to model this kind of variability using constructs like variation part, variant part, and select logic. This allows you to manage options **within a single model** instead of duplicating your architecture multiple times.

Why Model Variants in SysML?

Systems rarely exist in only one form. Most real-world architectures support multiple versions — differing in features, functions, or physical components. Modeling these **variants** directly in SysML helps you:

- **Avoid duplicating** models for each version
- Maintain a **single source of truth** with embedded flexibility
- Enable consistent **traceability** across configurations
- Plan for **reuse**, customization, and adaptation early in the design phase

Variability modeling turns static architectures into dynamic, configurable system families.



What Is Variability in Systems?

Why are we even talking about variability? Because in the real world, systems don't come in just one flavor.

Let's say your organization builds drones:

- Some customers want delivery drones
- Others want long-range surveillance drones
- Some want both, with hot-swappable modules

If you try to maintain a separate model for each configuration, your modeling effort quickly explodes into duplication, inconsistencies, and maintenance nightmares.

Instead, SysML v2 lets you **embed variability directly into the model** — using variation constructs. This means you define one model, with clean and explicit **variation points** where differences occur. Each configuration is then just a different **selection** — not a different model.

It gives you the benefits of:

- Reuse
- Maintainability
- Full traceability across all configurations
- And it lets you support **product line engineering** without chaos

That's why variability modeling isn't a luxury — it's a necessity for scalable MBSE.

Product Lines vs. Single System Models

A **single system model** describes one specific configuration — useful for prototyping or focused analysis. A **product line model** captures a whole family of configurations using variation and variant constructs — enabling reuse and flexibility.

Aspect	Single System Model	Product Line Model
Scope	One fixed system	Multiple configurations
Reuse	Low	High — shared architecture
Complexity	Grows linearly	Managed through variation logic
Adaptability	Hard to adapt	Easy to adapt using variants
Best Used For	Proof-of-concept, early demo	Long-term product architecture

What Is Variability in Systems?

Let's clearly distinguish two modeling approaches:

The **single system model** vs. the **product line model**.

- A **single system model** is focused — it represents one configuration. It's great when you're prototyping, validating a concept, or doing detailed analysis on one use case. But as soon as you need to manage different customers, use cases, or hardware combinations — it falls short.
- A **product line model**, on the other hand, introduces **variation points** — places in the model where you define alternatives. You use elements like variation part, variant part, and selection logic to model a whole family of systems. This supports **mass customization** without duplication.

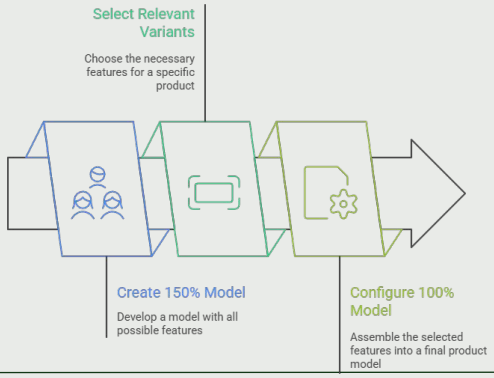
The benefits?

- Less model maintenance
- Greater consistency
- Reusable components
- Easier updates and extensions

This is what we'll build today: a **configurable drone system** that can represent multiple mission configurations from a single, coherent model.

The 150% Model in Variability

The 150% model is a system model that contains all possible features, functions, and configurations in one superset representation. It goes beyond a single product configuration by including every structural and behavioral option, even though no single product will use all of them at once. By applying variation points and selecting variants, individual 100% product models are derived from the 150% base. This approach ensures consistency, reuse, and adaptability across a product family.



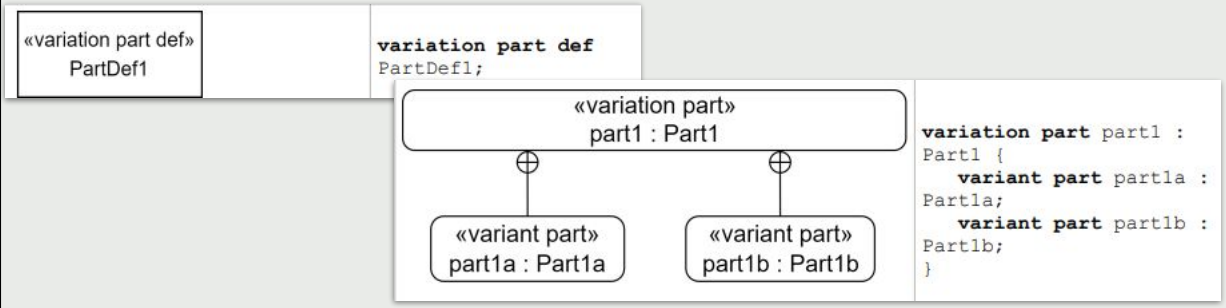
When we talk about a “150% model,” we mean a model that is intentionally bigger than any one system we would ever build. It has every possible feature and option included, whether or not they appear together in a real product. From this superset, we then configure actual products by selecting only the relevant variants, producing what we call the 100% model. This method is common in product line engineering because it prevents duplication, keeps traceability intact, and allows us to manage families of systems in one place. Instead of juggling multiple separate models for each variant, we keep everything in a single coherent structure and tailor it down when creating specific configurations.

What Is variation in SysML v2?

In SysML v2, a variation identifies a **decision point** in the model where multiple alternatives may be applied. It enables a single model to describe multiple configurations using **structured options**.

- A variation acts as the anchor for one or more variation parts or variation items
- Each variation holds possible variants, which are selected based on design intent
- Used to represent optional features, alternate designs, or product line diversity

variation is the foundation for building flexible, configurable system models.



Core Variability Elements

Let's start with the core concept: variation.

In SysML v2, a **variation** element declares a point in the system model where **alternatives are expected**. It doesn't define the alternatives itself — instead, it groups the possible variants through variation parts or variation items.

Think of variation as a **switchboard** — a placeholder that says,

"Here's where you'll decide between options like Camera A, Camera B, or no camera at all."

You don't have to duplicate your architecture — you just point to different variants depending on your configuration.

This becomes extremely powerful when your product must support:

- Optional modules (e.g., battery, sensors)
- Swappable payloads (e.g., thermal vs. optical camera)
- Alternate control strategies (e.g., manual vs. autonomous flight)

Every variant stems from a clearly defined variation. You define it once, and plug in different realizations depending on mission or customer needs.

This concept is what makes **model-based product line engineering** possible.

Understanding variation part and variation item

variation part and variation item define **where** variation occurs in the system:

- **variation part** → introduces variability in the system's structure (e.g., interchangeable sensors or payloads)
- **variation item** → introduces variability in data, flows, or behavioral elements (e.g., telemetry formats, mission types)

Each variation is defined at the model level, and its options are expressed through variant part or variant item.

These elements help express **"what might change"** in your system.

Core Variability Elements

When you want to model variability in SysML v2, you start by defining **where** the variability will occur.

SysML provides two primary variation mechanisms:

- A **variation part** defines a variation point in the **structure** — for example, where you might switch between different camera modules, batteries, or communication hardware. These typically replace physical or logical components.
- A **variation item** defines a variation point in **non-structural elements** — such as signal types, data messages, or logical behaviors like mission flows. This is great for cases like "Manual Control" vs. "Autonomous Mode."

These elements don't list their variants directly — instead, they reference one or more **variant part** or **variant item** definitions that specify the options available.

Think of it this way:

- variation part says: "Here's a slot for a component that could change."
- variation item says: "Here's a behavior or property that could vary."

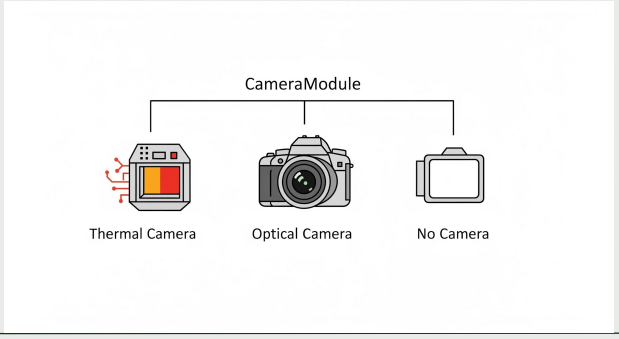
They allow you to build **adaptable system skeletons** that remain valid across many configurations.

Defining variant part and variant item

variant part and variant item specify the concrete alternatives that can populate a variation point:

- **variant part** → defines a structural alternative (e.g., ThermalCamera, OpticalCamera, NoCamera)
- **variant item** → defines a data, flow, or behavior alternative (e.g., ManualFlight, AutoPilot)

Each variant is declared using a variant part def or variant item def, and is linked to its associated variation through select. These define **"the available options"** for each configurable aspect of the system.



Core Variability Elements

Once you've declared a variation part or variation item, the next step is to define the possible options — these are your **variants**.

- A **variant part** is a specific alternative for a structural element. For example:
  - variant part ThermalCamera
  - variant part OpticalCamera
  - variant part NoCamera

These are all candidates for filling the same variation slot — like plug-in modules.

- A **variant item** works the same way, but it applies to behaviors, flows, or data properties — things like flight control logic or telemetry formats.

Each variant part or variant item is backed by its own definition (variant part def, variant item def), and in the final model, **only one is selected** per configuration.

This keeps your model both:

- Flexible (many possibilities)
- Precise (only one used at a time per config)

You now have both ends of the equation:

- The **variation** says "here's a choice to be made"
- The **variants** define "here are the valid options"

How Variant Selection Is Realized in SysML v2

SysML v2 does not use a select keyword — instead, variant selection is realized through **model usage** and **binding**.
You define a variation part or variation item, and in each configuration, you instantiate exactly one of its associated variant part or variant item.

- Variants are **declared** under the variation point
- One variant is **used** in each system configuration
- The selection is handled at the **usage level** — not through a keyword

This makes the model declarative and flexible — selections emerge through context, not imperative syntax.

Core Variability Elements
Let's clarify something important:
While we often say "selecting a variant," SysML v2 doesn't have a literal select keyword.
Instead, the selection is **realized by how the model is used**.
Here's how it works:

- You declare a variation part in your system — that's the variation point.
- You define several variant parts underneath it — these are your options.
- When you build a **specific configuration**, you choose **which variant to use** by referencing it in the model structure.

For example:

- You don't write something like select OpticalCamera
- Instead, your configuration instantiates DroneWithOpticalCamera — and that instantiation **uses** the OpticalCamera as the variant under CameraModule.

So while "selection" is part of the mental model, it's implemented as **model configuration and usage binding** — not as a SysML keyword.
This keeps the language clean and composable, and aligns with the declarative modeling style.

Modeling Structural Alternatives

To model structural variability, use variation part to represent a **replaceable component**, such as a sensor or payload.
Then define multiple variant part options under that variation, each with its own structure.
Example: variation part PayloadModule

- variant part ThermalCamera
- variant part OpticalCamera
- variant part EmptyBay

Each drone configuration instantiates one variant based on mission needs.

Applying Variability to Structure and Behavior
Let's make this with a drone example.
Imagine your drone has a slot where different payloads can be attached — such as:

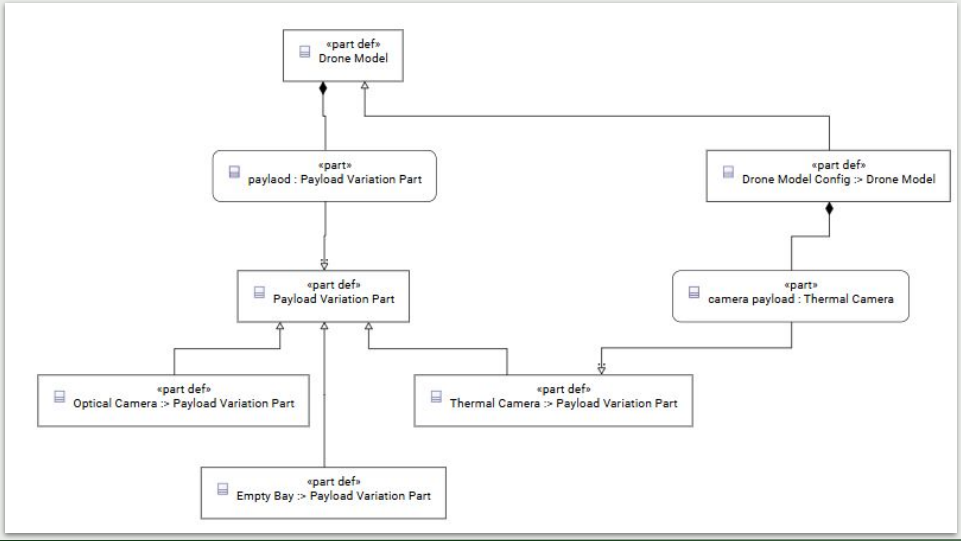
- A thermal camera for night operations
- An optical camera for high-resolution mapping
- Or an empty bay when you need to minimize weight

In SysML v2, you would define a **variation part** called PayloadModule. This acts as the placeholder — the slot in your structure where a variant will go.
Then you define your **variant parts**:

- ThermalCamera
- OpticalCamera
- EmptyBay (yes, absence is also a valid variant)

When building a particular drone configuration, you instantiate the structure and include the **variant part** that matches the mission.
This way, you don't copy your whole model for each variant — you just reuse the shared architecture and **swap the payload**.
This is clean, traceable, and scalable — perfect for product line engineering.

Modeling Structural Alternatives



Modeling Behavioral Alternatives

Behavioral variability is modeled using variation item to define a flexible point in a mission sequence.
You then define different variant item options to represent alternative behaviors or logic flows.
Example: variation item MissionType

- variant item SurveillanceFlow
- variant item DeliveryFlow
- variant item EmergencyLandingFlow

Each system configuration instantiates the mission behavior that fits its operational profile.

Applying Variability to Structure and Behavior

Structural changes are only half the story — **behavioral variability** is just as important.
Let's say your drone can perform:

- Surveillance missions
- Package deliveries
- Or emergency landings

These are **not just structural differences** — they involve entirely different sequences of behavior.
In SysML v2, you model this by declaring a variation item like MissionType.
This becomes your behavioral variation point — a placeholder where multiple logic paths could occur.
Then you define variant items, such as:

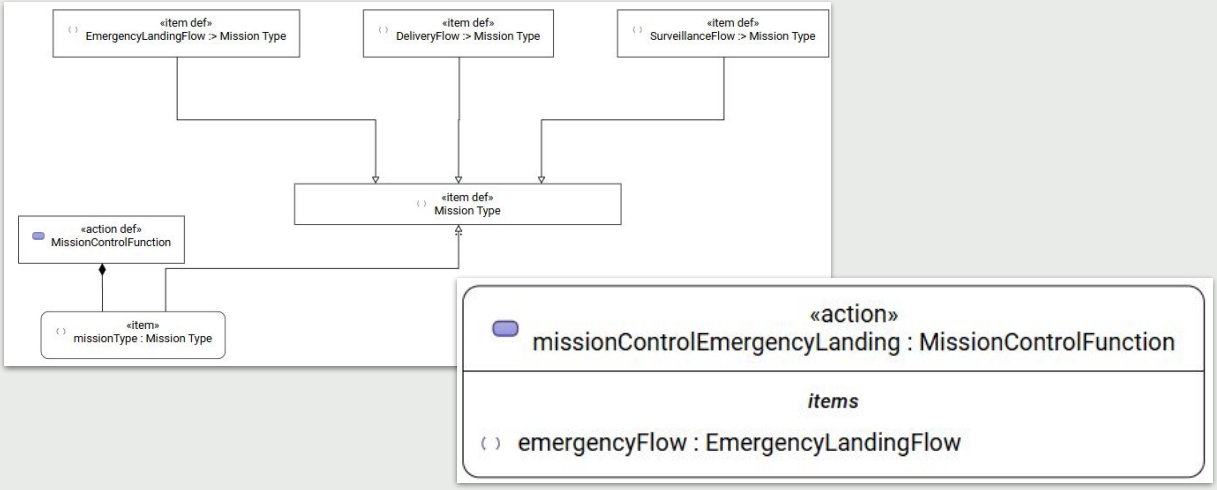
- SurveillanceFlow — a behavior tree with camera scanning and hover logic
- DeliveryFlow — a flow that includes waypoint navigation and drop-off logic
- EmergencyLandingFlow — a fallback mission with power prioritization and descent

When you instantiate your system for a specific mission, you include one of these variant items in place of the variation point.
This allows your **mission logic to flex** — without rewriting the entire behavioral model.
Behavioral variability is especially useful when:

- The same hardware supports different software modes
- The mission logic changes per customer
- You want to simulate or test different operational scenarios

And just like with structure — this approach gives you **clarity, reuse, and configuration control**.

Modeling Behavioral Alternatives

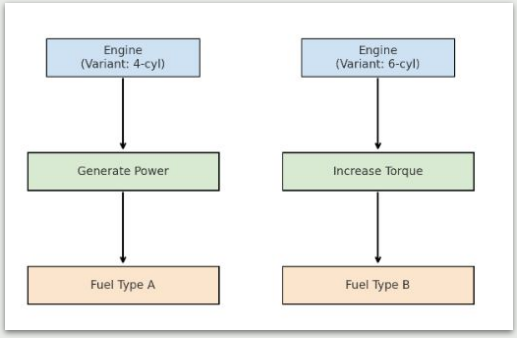


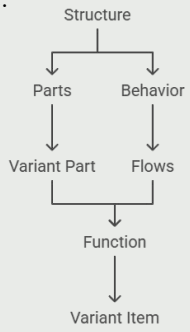
Linking Variants to Parts and Actions

Variants should integrate seamlessly with system architecture and logic:

- **variant part** instances are embedded within the structure of the system
- **variant item** instances participate in behaviors, interactions, or flows
- Variants can be traced to specific functions, constraints, or mission requirements

Linking structure and behavior ensures each configuration is both complete and testable.





Applying Variability to Structure and Behavior

Defining variants is powerful — but their value comes when you **integrate them into the model's structure and behavior.**

Here's how that works:

- When you instantiate a **variant part** like ThermalCamera, it doesn't just sit isolated — you plug it into the structure (e.g., into PayloadModule), and it interacts with other parts (like the Flight Controller or Power Unit).
- Similarly, when you use a **variant item** like DeliveryFlow, it becomes part of your system's behavior — controlling how the drone navigates, drops payloads, and ends the mission.

To ensure correctness, you should:

- Trace **each variant** to the **requirement or use case** that demands it
- Link **behavioral variants** to the **actions or functions** they affect
- Validate that **structural variants** support the behaviors and interfaces they connect to

This integrated traceability allows:

- Complete system validation for each configuration
- Verification that structure and behavior are always aligned
- Easier debugging and simulation of alternate versions

A good variability model is not just a set of options — it's a **well-wired configuration machine**.

Diagram View — One System, Multiple Configurations

A single SysML v2 model can support multiple drone configurations through variation constructs.

- Example Configurations:
- **Surveillance Drone:** ThermalCamera + SurveillanceFlow
- **Delivery Drone:** OpticalCamera + DeliveryFlow
- **Emergency Mode:** EmptyBay + EmergencyLandingFlow

Each configuration:

- Uses the same base model
- Binds different variants at variation points
- Remains fully traceable and reusable

From a **single system model**, you can generate **multiple configurations**, each tailored to a specific mission — without duplicating your architecture. Here's how the combinations look:

- A **Surveillance Drone** might use:
  - ThermalCamera as the variant part
  - SurveillanceFlow as the variant item
- A **Delivery Drone**:
  - Swaps in OpticalCamera
  - Uses DeliveryFlow behavior
- An **Emergency Mode Drone**:
  - Uses EmptyBay (lightweight config)
  - Falls back on EmergencyLandingFlow

All these share the same:

- Power system
- Navigation core
- Communication structure

But they differ only where **variation is explicitly allowed**.

This design pattern gives you:

- Configurable flexibility
- Code and model reuse
- Better lifecycle management

It's not about building *more* models — it's about building **smarter** ones.

Tips for Clean Variant Modeling

✅ Keep your variation points focused — don't overuse variation part or variation item
✅ Group related variants under clear and descriptive names
✅ Avoid mixing structural and behavioral variation in one element
✅ Always link variants to use cases, requirements, or missions
✅ Use consistent naming for variant part def and variant item def
Good variability modeling keeps the system flexible **without clutter**.

As you build out variation in your drone model, here are a few practical tips to keep things clean and scalable:

- **Don't overdo it.** Only use variation part or variation item where there's actual need for configurability. Every variation point adds complexity.
- **Name clearly.** Group your variants under intuitive names like PayloadModule, MissionType, CommsStack — not generic ones like "Option1" or "VarX".
- **Separate structure from behavior.** A variant part should only handle hardware or composition changes. A variant item handles logic or data flow. Don't mix these.
- **Trace every variant.** Always connect a variant to a use case or mission scenario. This helps explain why it exists — and ensures it has test and verification backing.
- **Use consistent naming.** Stick to a clear pattern like ThermalCameraDef → ThermalCamera, SurveillanceFlowDef → SurveillanceFlow. This makes your model easier to read and navigate.

Clean modeling = better models, easier debugging, and happier reviewers.

Summary of Week 13

This week, you learned how to model configurable system architectures using SysML v2 variation constructs.

🔑 Key Concepts:

- variation part / variant part → for structural variability
- variation item / variant item → for behavioral or logical variability
- Configurations are realized by **instantiating variants** — no select keyword
- One model supports multiple drone types and missions
- Clean variation = reusable, traceable, and adaptable architecture

Variability transforms your model from a system to a **system family**.

Let's quickly review what we've covered this week.

- You now know how to declare **variation points** in your model using variation part and variation item.
- You define your **options** using variant part and variant item, and wire them in based on the configuration.
- Unlike a programming switch or config file, you don't "select" a variant with a keyword — instead, selection happens through how you **use and instantiate** model elements in your configuration.
- This approach lets you build one model that adapts to:
    - Delivery drones
    - Surveillance drones
    - Emergency fallback modes
    - And more

The power of SysML v2 variability is that it helps you:

- Avoid duplication
- Capture real-world flexibility
- Maintain rigorous traceability

QUESTION!