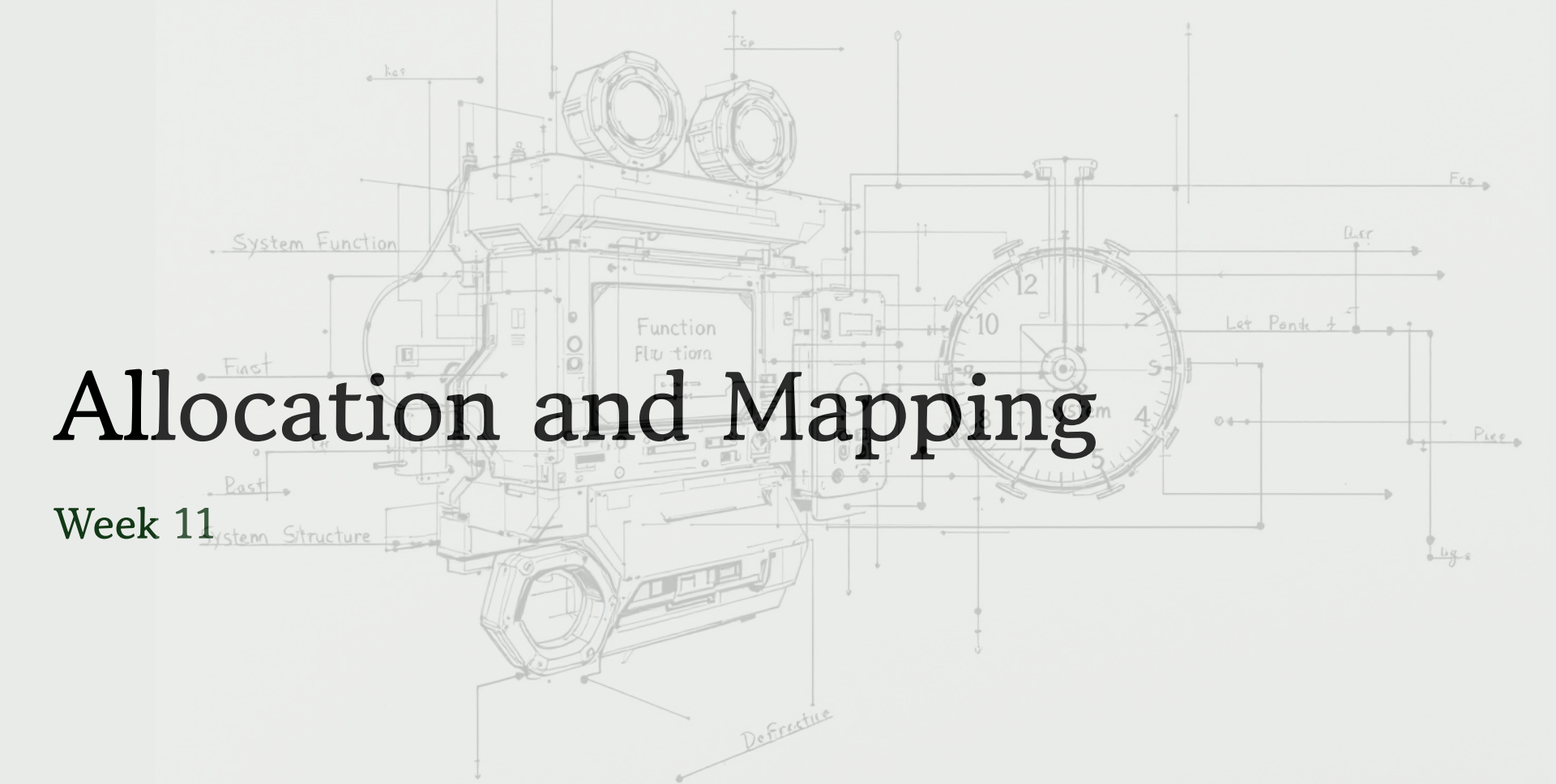



# Allocation and Mapping

## Week 11

# Allocation and Mapping

Bridge behavior, structure, and requirements by teaching allocation and traceability in SysML v2.
You will learn to build complete traceability chains and apply formal relationships to connect stakeholder intent to architecture and system logic.

# Why Allocation Matters in Systems Engineering

**Systems engineering requires connecting high-level intent with detailed system design.**
Allocation bridges the gap between:

- **What** the system must do (Needs and Requirements)
- **How** the system achieves it (Behaviors and Architecture)

Without proper allocation:

- It is difficult to show how the design meets stakeholder intent
- It is harder to ensure requirements are satisfied and verified
- Traceability for certification or compliance is incomplete

**Allocation makes models actionable — not just descriptive.**

# Traceability: From Stakeholder Intent to Architecture

**Traceability connects all levels of the system model:**

- **Stakeholder Need** — captures intent
- **Requirement** — formalizes what the system must achieve
- **Behavior** — defines how the system will operate
- **Structure** — identifies which parts perform which behaviors
- **V&V** — defines how compliance will be verified and validated

**Complete traceability enables:**

- Understanding how architecture satisfies intent
- Supporting certification and compliance
- Managing impact of changes
- Improving communication across stakeholders

# What Is Behavior-to-Structure Mapping?

**Behavior-to-Structure mapping shows which part of the system performs which behavior.**
In SysML v2, we allocate behaviors (e.g., actions, activities, state machines) to structural elements (part usages).
This creates an explicit link between the system's logic and its physical architecture.
Example:

- "NavigationControl" activity is executed by the "FlightController" part
- "BatteryMonitoring" state machine is hosted by the "PowerModule" part

**This allocation ensures that system logic is implemented by actual system components.**

# How to Allocate Behaviors to Parts (Part Usages)

**Behavior allocation uses a formal relationship to connect a behavior element to the part that performs it.**

In SysML v2:

- Behaviors (e.g., action, state) are allocated using the <<allocate>> relationship
- The target is typically a **Part Usage** in the structural model
- This defines **execution responsibility** — who does what

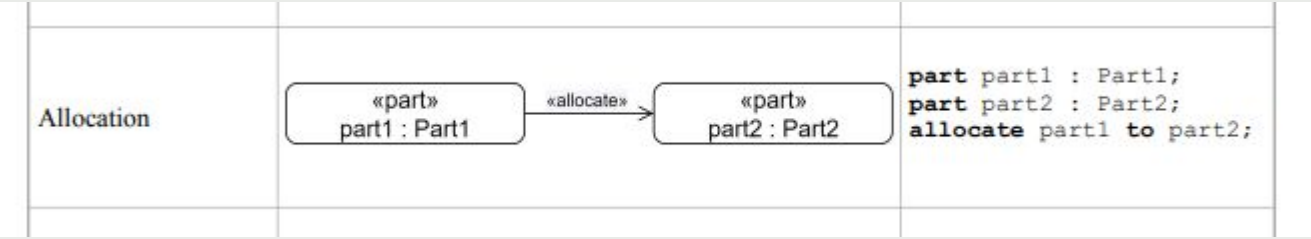
**This relationship supports traceability and consistency across system layers.**

# Good Practices for Behavior Allocation

**Effective allocation improves model clarity, traceability, and future implementation.**

- ✔ Allocate every major behavior to a responsible part
- ✔ Keep allocation one-to-one where possible — avoid ambiguous mappings
- ✔ Use meaningful naming for behaviors and parts
- ✔ Group related behaviors under the same controller or subsystem
- ✔ Visualize allocations early to identify gaps or overloads

**Allocation is not just traceability — it defines design intent.**



# Allocating Requirements to System Elements

**Requirement allocation links abstract system goals to concrete design elements.**

In SysML v2, a requirement can be allocated to:

- A **Part Usage** — indicating responsibility for implementation
- A **Behavior** — indicating the function that satisfies it
- A **Constraint** — if satisfaction is based on calculated performance

**This creates a direct, traceable path from Requirement to System Design.**

# Tracing Need → Requirement → Architecture → V&V

**A complete traceability chain connects system intent to implementation and verification.**

Typical trace path:

- **Stakeholder Need** — what the user wants.
- **Requirement** — what the system must do.
- **System Elements** (Parts or Behaviors) — how the system does it.
- **Test or Analysis** — how we confirm it is done correctly

**This enables end-to-end traceability and lifecycle assurance.**

# Example Trace Chain: From Need to Part Usage

**Scenario:**

The drone must operate safely in low-visibility conditions.

Trace chain:

- **Stakeholder Need:** “Support safe operation in foggy environments”: refine by
- **Requirement:** “Drone shall maintain position within ±1 m under limited GPS visibility”: satisfy by
- **Behavior:** HoverControl activity : allocates to
- **Part Usage:** FlightController

**This complete chain shows who implements what, and why.**

# Good Practices for Requirement Traceability

**Clear traceability improves system clarity, auditability, and change management.**

- ✔ Use refine to connect Needs to Requirements
- ✔ Use satisfy to connect Requirements to Parts or Behaviors
- ✔ Use verify to connect Requirements to Test Cases or Analyses
- ✔ Keep relationships explicit and queryable in the model
- ✔ Avoid tracing requirements to too many elements — maintain clarity

**A clean trace model supports compliance, reuse, and impact analysis.**

# What Is ‘refine’ in SysML v2?

**The refine relationship connects an abstract Need to one or more detailed Requirements.**
It is used when:

- A **Stakeholder Need** must be elaborated into one or more measurable, actionable Requirements
- We want to trace high-level intent down into system logic

**refine clarifies how the system definition originates from user intent.**

**Example:**

- The need: “Support safe operation in foggy environments”
- Refined requirement: “Maintain position within ±1 m under degraded GPS signal”

# What Is ‘satisfy’ in SysML v2?

**The satisfy relationship links a Requirement to the system element responsible for fulfilling it.**
It is used to:

- Show which **Part Usage** or **Behavior** implements the requirement
- Define clear ownership and accountability
- Enable automatic trace checks and verification planning

**satisfy answers the question: who or what makes this requirement true?**

**Example:**

- Requirement: “Maintain heading within ±5° in wind”
- Part: flightController
- Behavior: headingControl action

# What Is ‘verify’ in SysML v2?

**The verify relationship connects a Requirement to the test, analysis, or review used to confirm it.**
It is used to:

- Show how a Requirement will be **validated or confirmed**
- Link to **Test Cases**, **Simulations**, or **Constraint Checks**
- Enable traceability from requirement to verification evidence

**verify closes the loop — proving that the system does what it’s supposed to do.**

**Example:**

- Requirement: “Maintain flight for 30 minutes”
- Constraint: endurance >= 30
- Calculation: computes endurance based on battery and payload
- Verification: we create a verify link from the requirement to the constraint model

# Example: Using refine, satisfy, verify in a Traceability Chain

**Scenario:** Ensure drone remains stable in wind.
Traceability Chain:

- **Stakeholder Need:** “Maintain safe flight in moderate wind”

refine

- **Requirement:** “Drone shall hold heading within ±5° in 10 m/s wind”

satisfy

- **Behavior:** headingControl action

allocate to

- **Part Usage:** flightController

verify

- **Constraint Check:** abs(desiredHeading - actualHeading) <= 5°

**This chain shows intent, implementation, and how we confirm correctness.**

# Practice: Build a Full Traceability Chain

**Objective:** Model a complete trace from a Stakeholder Need to a Part Usage — using refine, satisfy, allocate, and verify.

**Scenario:** Stakeholder Need: “Ensure reliable braking in emergencies”

You will:

- Create a formal **Requirement**
- Define the **Behavior** that implements it
- Identify the responsible **Part Usage**
- Define a simple **Verification Constraint**
- Connect all with proper relationships in SysON

# Build Traceability Chains in Your Drone Model

**Objective:** Create full traceability chains in your Autonomous Drone System using real stakeholder needs and requirements.

Your task:

- Choose at least **2 Stakeholder Needs** from your model
- Refine each into **1 or more Requirements**
- Define the **Behaviors** that satisfy those requirements
- Allocate the behaviors to appropriate **Part Usages**
- Add at least one **Verification Element** (test, constraint, or calc)
- Use refine, satisfy, allocate, and verify relationships

# Summary of Week 11

This week, we learned how to:

- **Allocate Behaviors** to Parts using allocate
- **Connect Requirements** to System Elements using satisfy
- **Verify Requirements** using tests, constraints, or analysis via verify
- **Refine Stakeholder Needs** into Requirements using refine
- Build full **traceability chains** from intent to implementation

**Traceability turns models into evidence — ready for review, testing, and compliance.**

# QUESTION!