

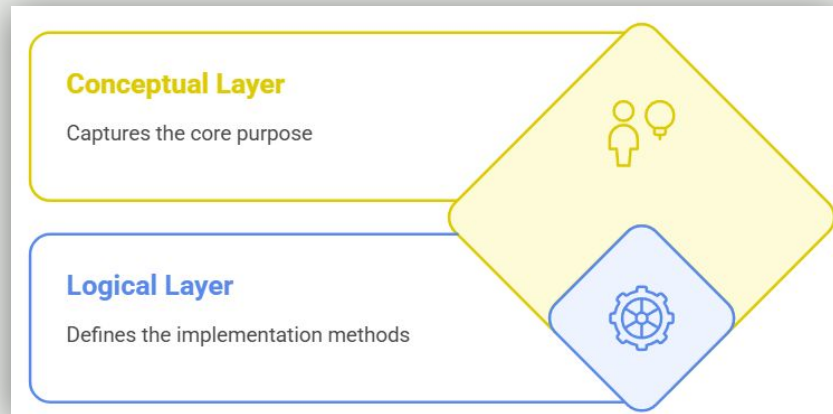
Modeling Stakeholder Needs and Concerns

Week 04

Modeling Layers in MBSE

In MBSE, we model systems by separating concerns into different levels of abstraction. This helps us focus first on understanding the *problem* before designing the *solution*. We call these levels the **Conceptual Layer** and the **Logical Layer**.

- The **Conceptual Layer** captures the *why* — stakeholders, needs, context, and use cases.
- The **Logical Layer** defines the *how* — requirements, architecture, behavior, and constraints.
- Layers are not steps, but **perspectives** to structure thinking and modeling.
- A well-built system model flows smoothly from **Conceptual** to **Logical**.
- Today's session begins our journey in the **Conceptual Layer**.

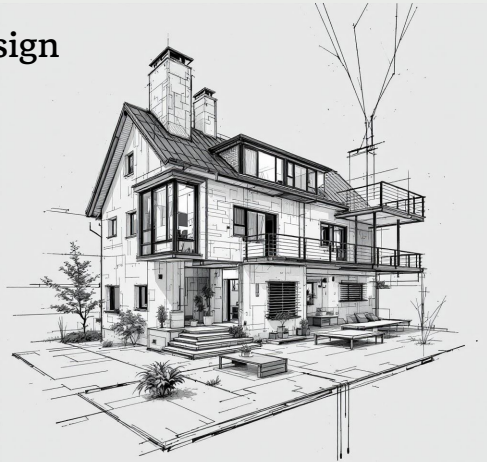


Conceptual vs Logical

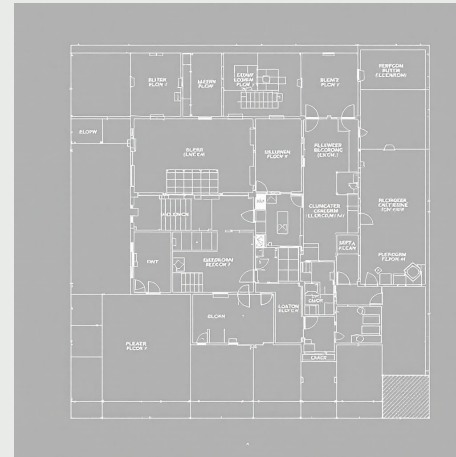
Imagine you're designing a building. First, you sketch its purpose and layout — not yet the wiring or materials. Only after that do you plan the structure in detail. MBSE works the same way.

- The **Conceptual Layer** is like an **architect's blueprint** — it captures intent, function, and usage.
- The **Logical Layer** is like an **engineering plan** — it details how things are connected and built.
- Starting with the Logical Layer too early leads to **fragile models** and rework.
- Working top-down ensures **clarity, traceability, and alignment** with stakeholder needs.

Conceptual Design

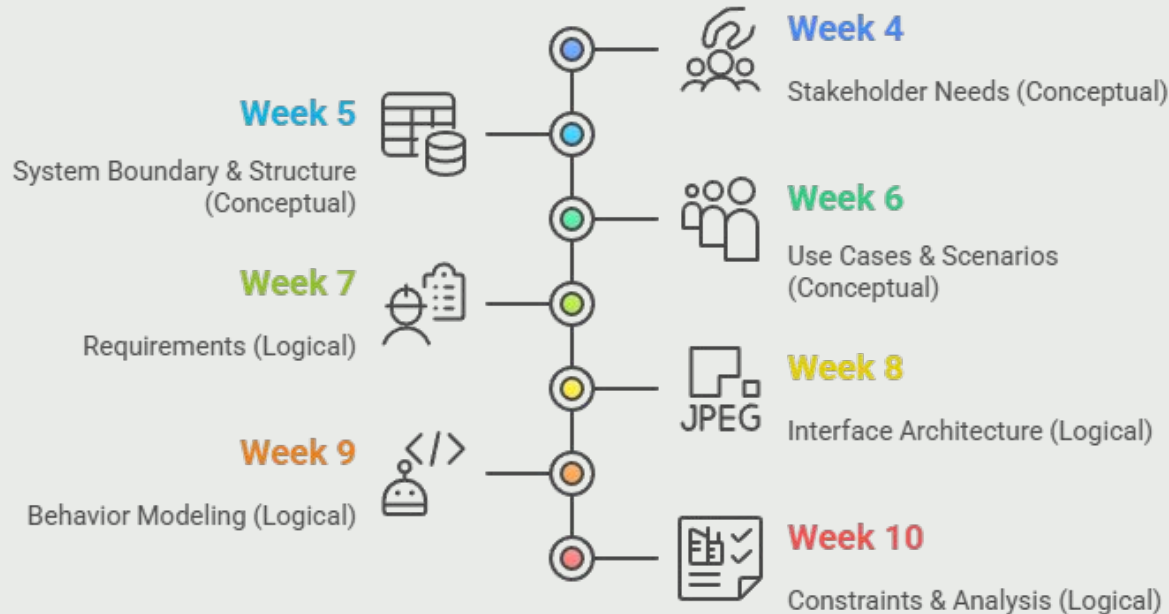


Logical Design



Journey Across the Layers

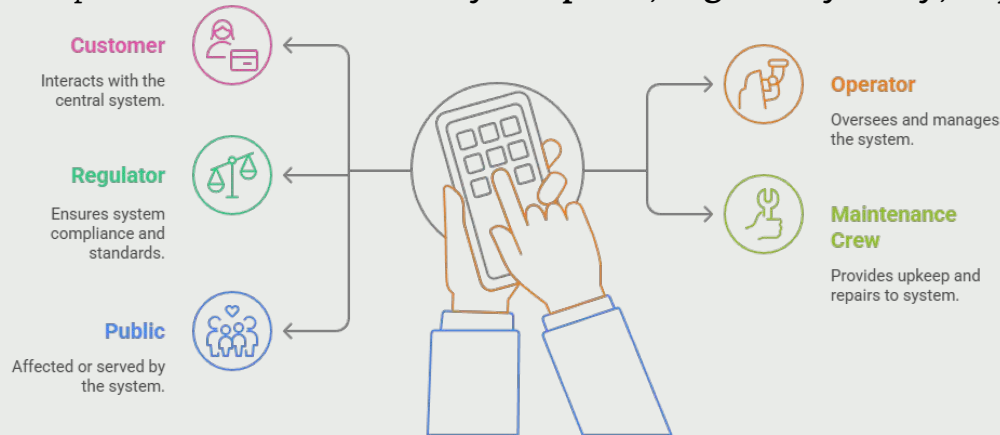
In this course, we'll model systems by moving through the Conceptual Layer and then the Logical Layer. Each week builds upon the last — just like real-world MBSE projects. This roadmap shows how your system model will evolve over time.



Identifying Stakeholders

Before modeling a system, we must understand *who* is involved and *who* is impacted. A stakeholder is any person, group, or organization that affects or is affected by the system. Identifying them early ensures we design a system that meets real-world expectations.

- Stakeholders include **customers, users, operators, maintainers, regulators**, etc.
- Think broadly: internal and external, direct and indirect involvement.
- A stakeholder can influence needs, usage, environment, or compliance.
- For our drone example: consider the **delivery recipient, regulatory body, support team**, and even **neighbors**.



Stakeholder Examples

Let's apply stakeholder thinking to a real-world example: an Autonomous Delivery Drone System. By mapping different roles and their impact, we can begin to see whose needs will shape our design. This exercise helps ensure no important voice is left out.

- **Customer** – The person or business ordering drone deliveries
- **Operator** – The user or staff managing drone deployment
- **Maintenance Crew** – Responsible for repairs and routine checks
- **Regulator** – Aviation authority ensuring flight and airspace compliance
- **Recipient / End User** – Person receiving the delivered item
- **Logistics Supervisor** – Oversees fleet operations and scheduling
- **Public / Neighborhood** – Affected by noise, privacy, and safety

Stakeholder Identification

Identifying stakeholders may seem easy, but missing or misclassifying them can lead to serious design gaps. Use these tips to be thorough and unbiased, and avoid common pitfalls in stakeholder discovery.

Tips for Identifying Stakeholders

- Start from **system boundaries**: Who inputs/outputs data, materials, decisions?
- Ask: *Who installs it? Who uses it? Who maintains it? Who approves it? Who complains if it fails?*
- Use **real-world analogies** — think of logistics, operations, public safety.
- **Interview or brainstorm** — cross-functional teams help spot missing roles.
- Document assumptions — “*We assume the operator is the same as the user*” may not always be true.

Stakeholder Identification

Identifying stakeholders may seem easy, but missing or misclassifying them can lead to serious design gaps. Use these tips to be thorough and unbiased, and avoid common pitfalls in stakeholder discovery.

Common Pitfalls to Avoid



- Focusing only on internal users (e.g., developers, operators)
- Ignoring passive stakeholders (e.g., the public, environment)
- Forgetting indirect influencers (e.g., marketing, legal, policy makers)
- Confusing **stakeholders** with **system functions** or roles in the architecture

Organizing Stakeholders

Not all stakeholders play the same role — some drive decisions, others are affected passively. Categorizing stakeholders helps us prioritize needs and understand who shapes the system the most.

This is essential for aligning the system's focus with its most critical influencers.

By Role:

- **Primary Stakeholders** – Directly use or manage the system
- **Secondary Stakeholders** – Indirectly affected (e.g., public, regulators)
- **Tertiary Stakeholders** – Peripheral interest (e.g., marketers, insurers)



By Influence:

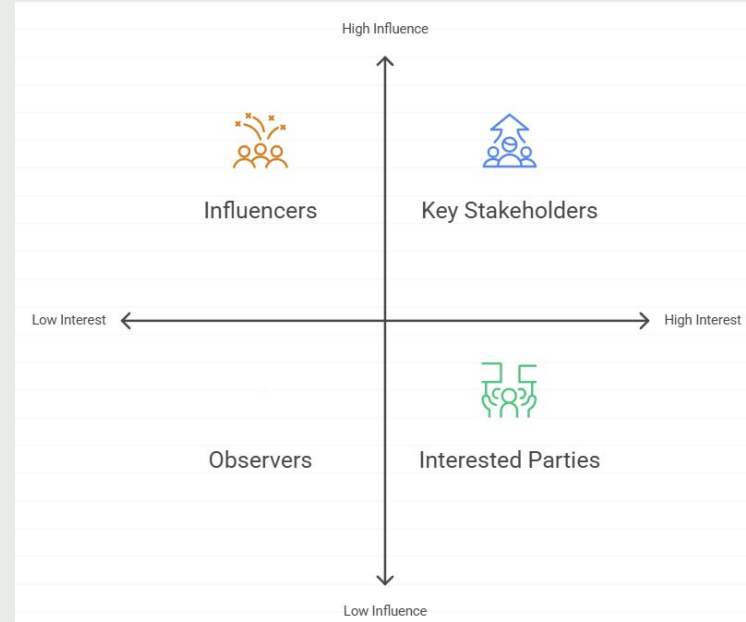


- **High Influence / High Interest** – Engage closely (e.g., Operator, Regulator)
- **High Influence / Low Interest** – Monitor (e.g., Policy Makers)
- **Low Influence / High Interest** – Inform (e.g., End Users, Public)
- **Low Influence / Low Interest** – Acknowledge, but minimal engagement

Stakeholder Influence vs Interest

Let's categorize drone stakeholders using a common systems approach: the **Influence vs Interest Matrix**. This tool helps you visualize where to focus your engagement, communication, and design attention.

Quadrant	Stakeholder	Notes
● High Influence / High Interest	Drone Operator, Airspace Regulator	Involved in operations and compliance — top priority
● High Influence / Low Interest	Logistics Supervisor, Policy Maker	Can shape policy or direction — keep informed
● Low Influence / High Interest	Customer, Recipient, Public	Impacted directly — ensure transparency and trust
● Low Influence / Low Interest	Insurance Agent, Local Vendors	Low risk, but should still be acknowledged



Capturing Stakeholder Needs

Stakeholder needs reflect the desired outcomes, conditions, or constraints that the system must fulfill. They help us understand what success looks like *from the user or beneficiary perspective*.

In MBSE, we model these needs using structured SysML v2 elements for traceability and clarity.

- **Stakeholder Needs** are informal expectations that guide system design.
- A good need should express *what* is wanted, not *how* it will be implemented.
- Needs often relate to *themes* like safety, reliability, or efficiency.
- Modeling needs helps translate abstract desires into formalized system requirements.

Modeling Needs Using SysML v2

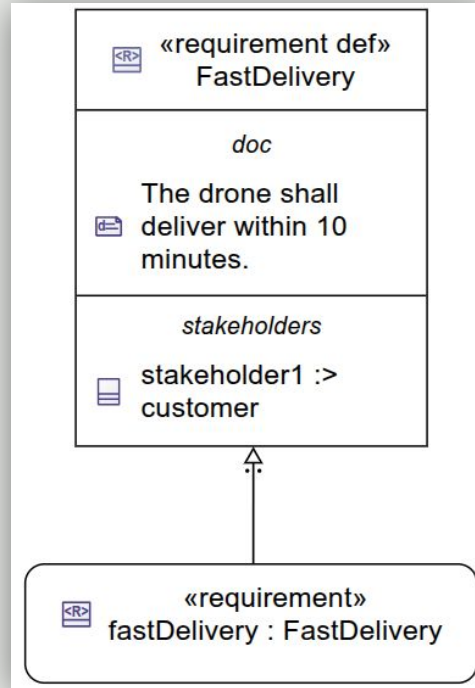
To model stakeholder needs formally in SysML v2, we don't treat them as just text — we make them part of the system model.

SysML v2 provides a structured way to represent each need as a reusable and traceable element.

- A **stakeholder need** is modeled as a **requirement element** in SysML v2.
- We use the keyword **requirement def** to define the structure of the need.
- The actual use of that need in a specific model context is written as **requirement**.
- These elements can include a **stakeholder property** to link the need to its origin.
- Representing needs this way ensures they are traceable, reusable, and integrated into the system model.

Modeling Needs Using SysML v2

Example of Graphic Notation for 'requirement def' and 'requirement with their relationship.

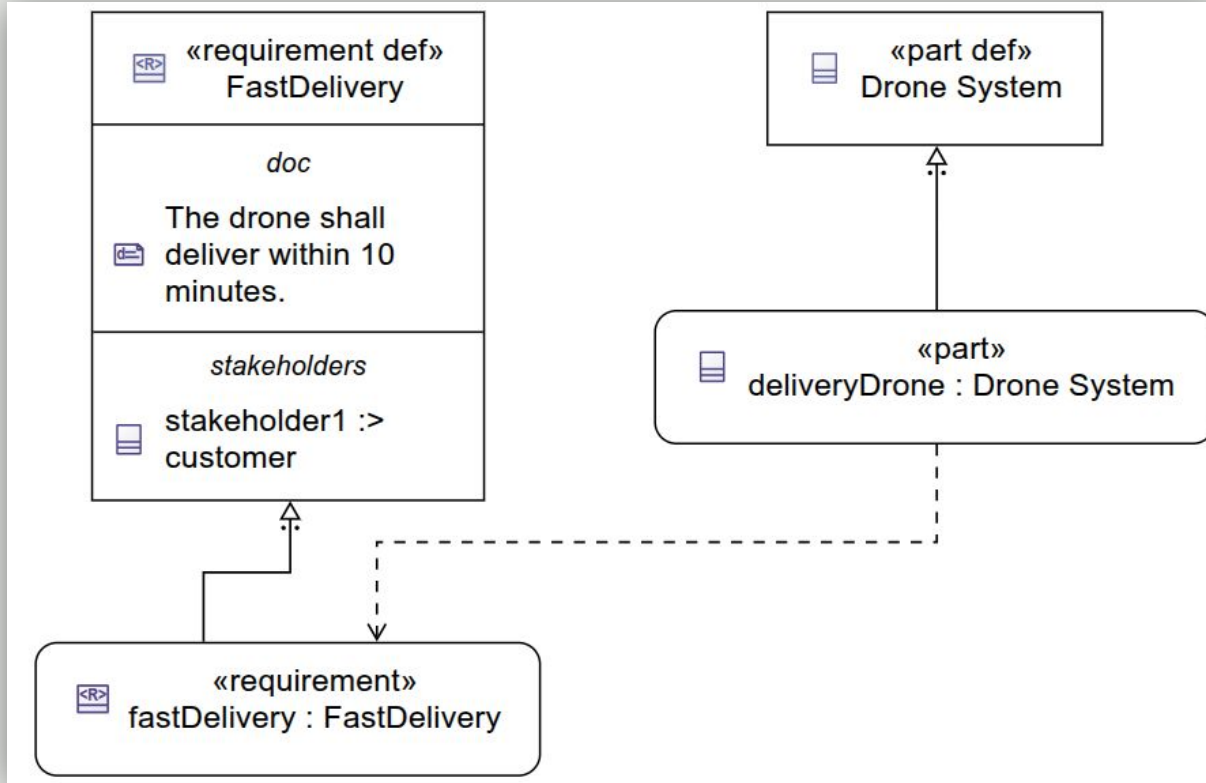


‘requirement def’ and ‘requirement’

In SysML v2, every modeling element has two forms: **definition** and **usage**. This pattern applies to requirements as well — and understanding the difference is key to building modular, scalable models.

- **requirement def** defines the reusable structure of a stakeholder need.
→ Think of it as the *template* or *definition*.
- **requirement** is a usage or reference to that definition within a specific model context.
→ It allows us to reuse the same requirement definition across multiple views or systems.
- Use **requirement def** when you want to *declare* a stakeholder need once.
- Use **requirement** when you want to *use*, *allocate*, or *trace* that need in structure, behavior, or views.
- This separation enables **clean traceability, versioning, and multi-view modeling**.

‘requirement def’ and ‘requirement’



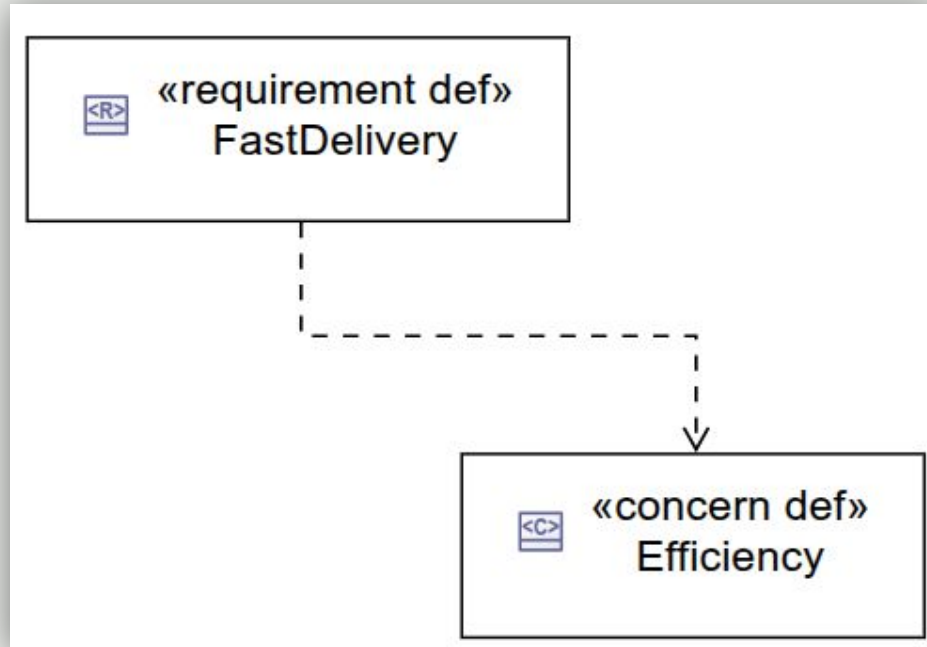
Modeling Stakeholder Concerns

Stakeholder needs often share common themes — like safety, cost, or environmental impact. In SysML v2, we model these themes as **concerns**, allowing us to group and manage needs more effectively.

- A **concern** represents a topic or quality the stakeholder cares about (e.g., *Safety*, *Efficiency*).
- Use **concern def** to define a reusable concern in the model.
- Use **concern** to reference or apply the concern within a specific context.
- A **requirement** (stakeholder need) can be linked to a concern using the **dependency** relationship.
- Concerns also include a **stakeholder property** to show who is most invested in that theme.

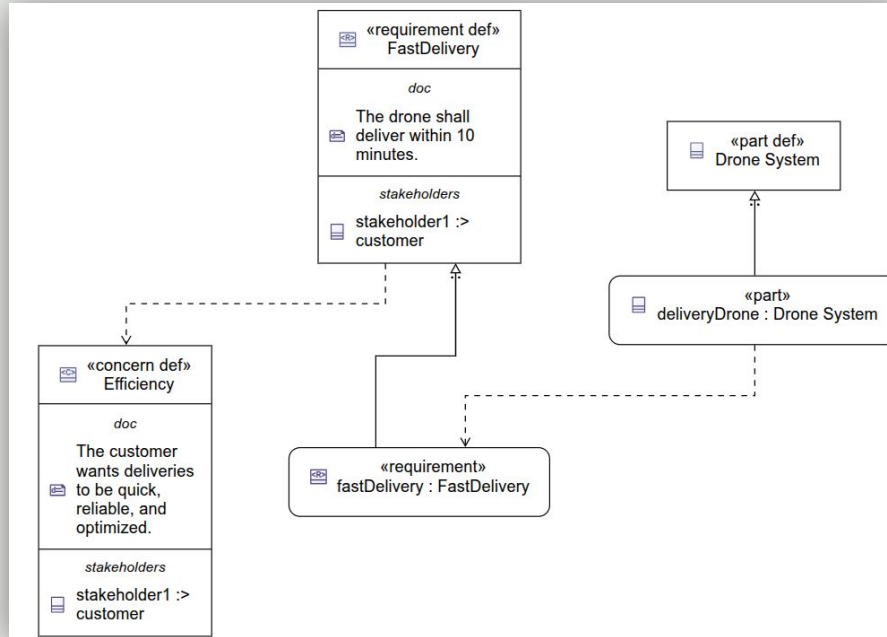
Modeling Stakeholder Concerns

Graphical notation to express that the requirement *FastDelivery* depends on the concern *Efficiency*.



Drone System Example

Let's look at how stakeholder concerns connect to specific needs in our drone delivery system. This mapping shows how abstract values like *efficiency* shape real, actionable expectations.

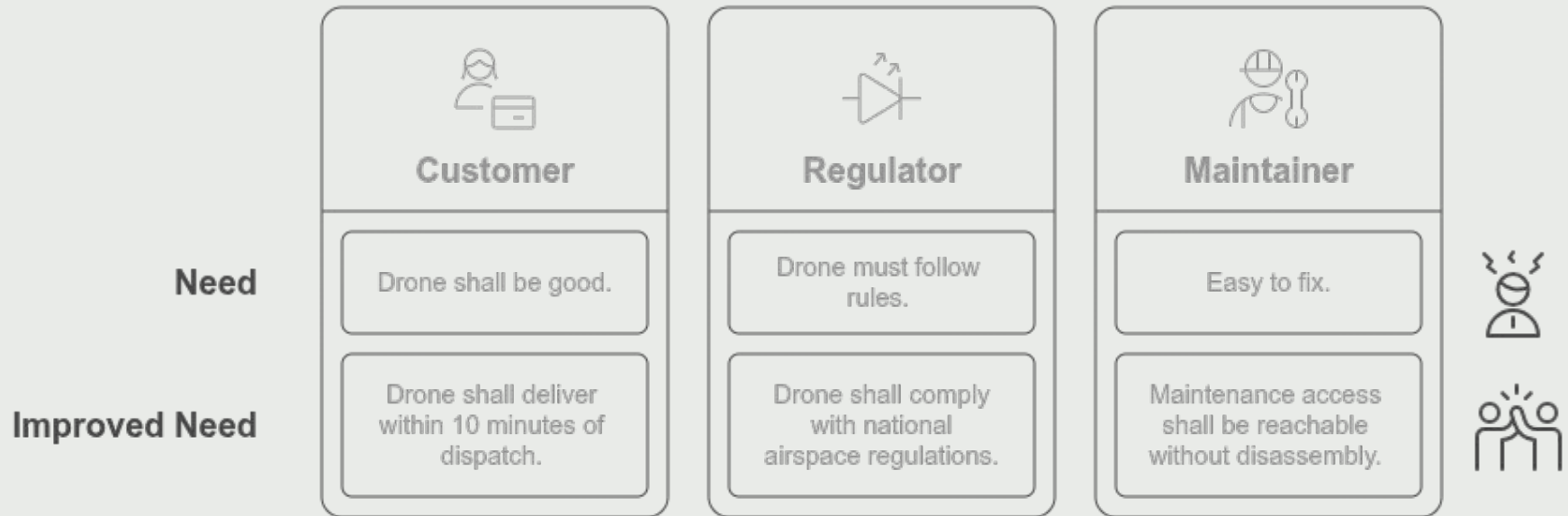


Writing Quality Stakeholder Needs

Not all needs are created equal. Poorly written needs create confusion, misalignment, and weak models. A good stakeholder need is clear, focused, and meaningful — without turning into a technical requirement.

- ✓ **Clarity** – The need should be easy to understand by both technical and non-technical audiences.
- ✓ **Singularity** – Each need should express **one single expectation or goal**.
- ✓ **Intent-Focused** – Describes *what* the stakeholder wants, not *how* to implement it.
- ✓ **Outcome-Based** – Linked to an effect, purpose, or value that the system provides.
- ✓ **Traceable to a Stakeholder** – Always model who the need came from.
- Avoid vague terms like “The system shall be good” or “The system shall be fast.”
- Avoid embedding multiple ideas in one need — e.g., “The drone shall be light and reliable and affordable.”

Writing Quality Stakeholder Needs






From Needs to Use Cases

Each stakeholder need leads to one or more system behaviors. We call these behaviors Use Cases — system interactions that fulfill a stakeholder goal. We introduce them now as a preview of Week 6, to show the path from *intent* to *action*.

- A **Use Case** describes a system interaction from the stakeholder's perspective.
- Each **need** should eventually be linked to a **Use Case** that fulfills it.
- Use Cases help answer: *What must the system do to meet this need?*
- At this stage, we don't model Use Cases — we simply **identify the interaction** it implies.
- We will model full Use Cases and Scenarios in **Week 6**.

From Needs to Use Cases

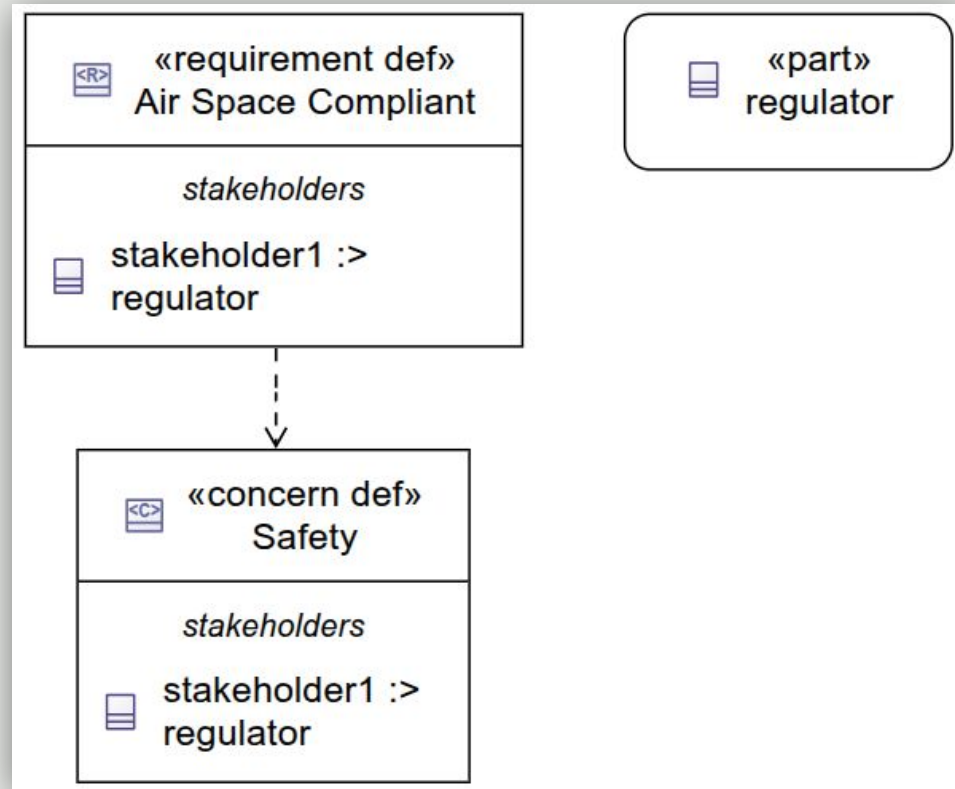
Needs	Leads to Use Case
“Deliver within 10 minutes”	Perform Delivery 
“Avoid restricted airspace”	Perform Navigation with Airspace Rules 
“Allow easy maintenance”	Perform Diagnostics 

Simple Practice

Let's build our first mini model to practice stakeholder-driven thinking. You'll create one stakeholder, one need, and one concern — and link them all together using SysML v2 elements.

- A **Stakeholder** is modeled using **part def** and its instance as a **part**.
→ We'll explore this in detail next week — for now, think of it as defining a role.
- A **Need** is modeled using **requirement def**.
- A **Concern** is modeled using **concern def**.
- The **stakeholder** property connects the stakeholder to both the need and the concern.
- Use a **dependency** relationship to connect the need to the concern.
- All of this will be done in SysON using graphical modeling.

Simple Practice



Autonomous Drone System



Now that you've practiced modeling a simple stakeholder-need-concern relationship, it's time to start your own system model. This is the first step of your course project — and it starts with modeling the people the system is meant to serve.

- Model **2–3 stakeholders** relevant to your **Autonomous Drone System**.
 - Use **part def** for each stakeholder and create a usage with **part**.
- Define at least **2–4 needs** using **requirement def**, each tied to a stakeholder using the **stakeholder** property.
- Create **1–2 concerns** using **concern def** — themes like Safety, Efficiency, or Usability.
- Connect needs to concerns using **dependency** relationships.
- Use graphical modeling in SysON — no need for textual syntax yet.
- This will be your **seed model**, which you'll grow throughout the course.

Summary of Week 4

This week, you began your modeling journey by capturing *who the system is for* and *what they care about*. You've laid the foundation for a system model that stays connected to stakeholder intent.

100

Learned to **identify and organize stakeholders**

100

Captured **stakeholder needs** using **requirement def**

100

Modeled **concerns** to represent values like Safety and Efficiency

100

Linked stakeholders, needs, and concerns through **traceable relationships**

100

Completed a **simple practice exercise** and began your **project model**



Next week, we shift focus to modeling the **system boundary and structure**

QUESTION!