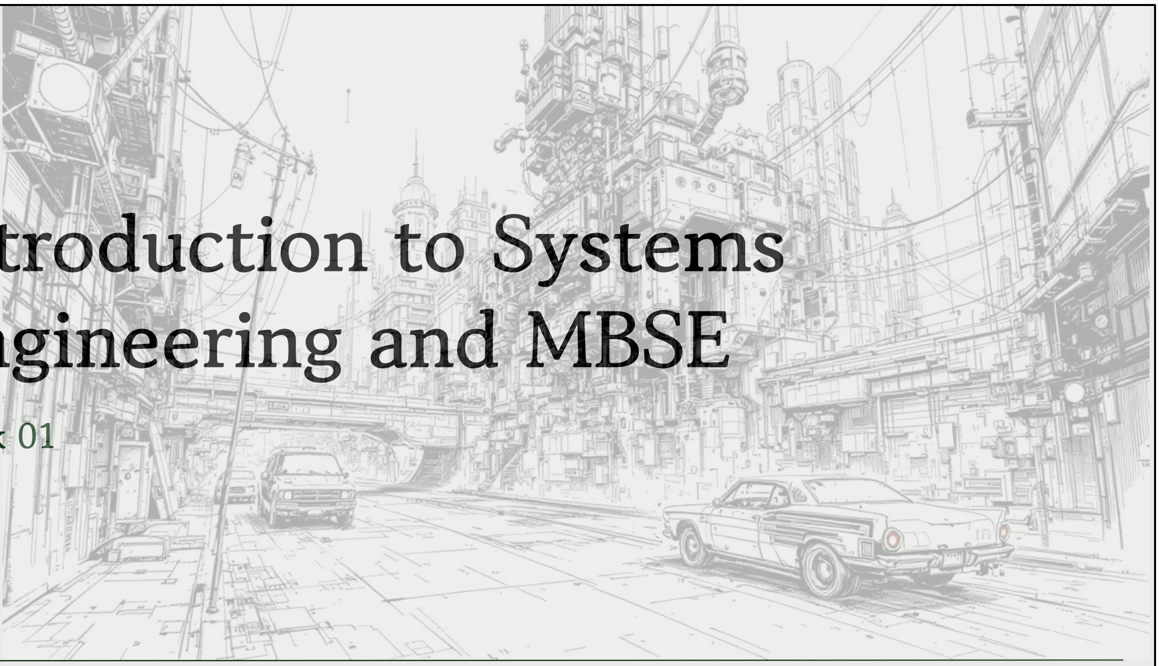
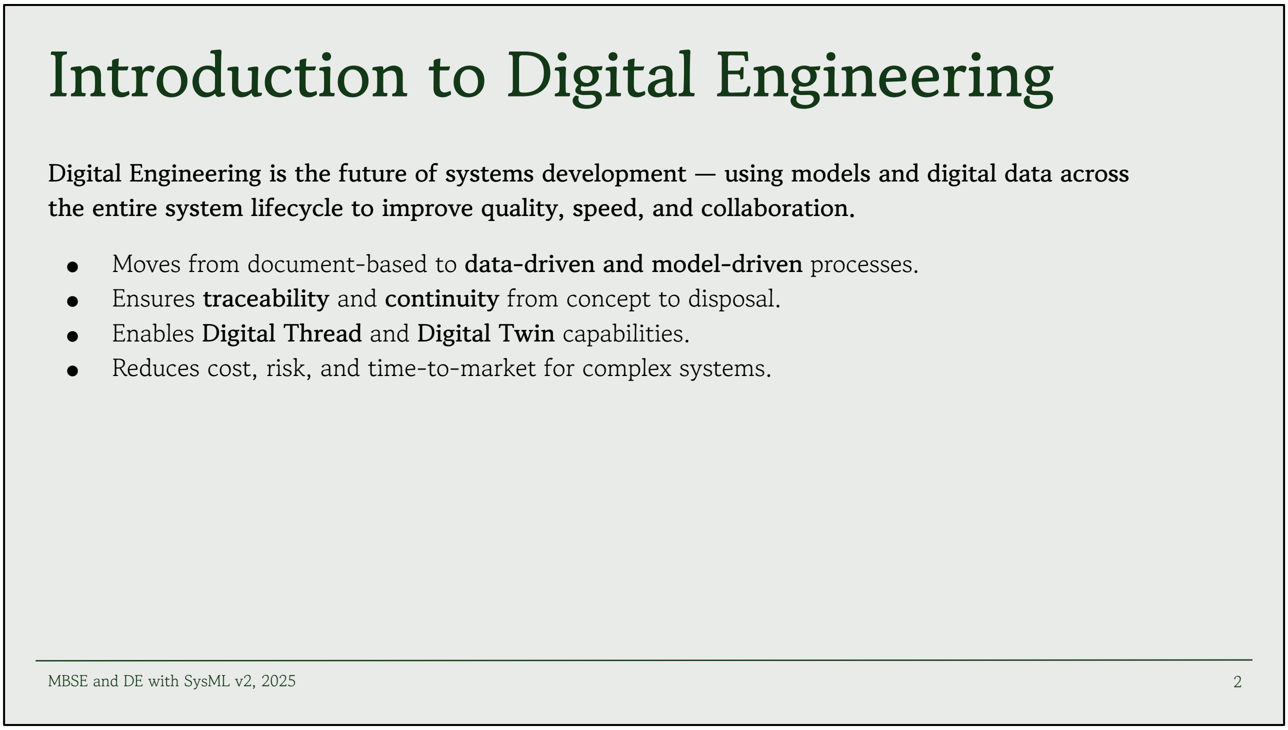


Introduction to Systems Engineering and MBSE

Week 01



Introduction to Digital Engineering

Digital Engineering is the future of systems development — using models and digital data across the entire system lifecycle to improve quality, speed, and collaboration.

- Moves from document-based to **data-driven and model-driven** processes.
- Ensures **traceability** and **continuity** from concept to disposal.
- Enables **Digital Thread** and **Digital Twin** capabilities.
- Reduces cost, risk, and time-to-market for complex systems.

Welcome to the beginning of your journey into **Digital Engineering** — the future of how we design, develop, and sustain complex systems.

In the past, engineering relied heavily on documents — requirements in Word files, designs in drawings, and test results in spreadsheets.

But today’s systems — from spacecraft to smart cities — are simply **too complex** and **too interconnected** for that approach to work reliably.

Digital Engineering represents a major shift: It means **replacing isolated documents** with **connected digital models and data** that stay alive throughout the system's entire lifecycle — from early concepts, through design and manufacturing, into operation, maintenance, and even decommissioning.

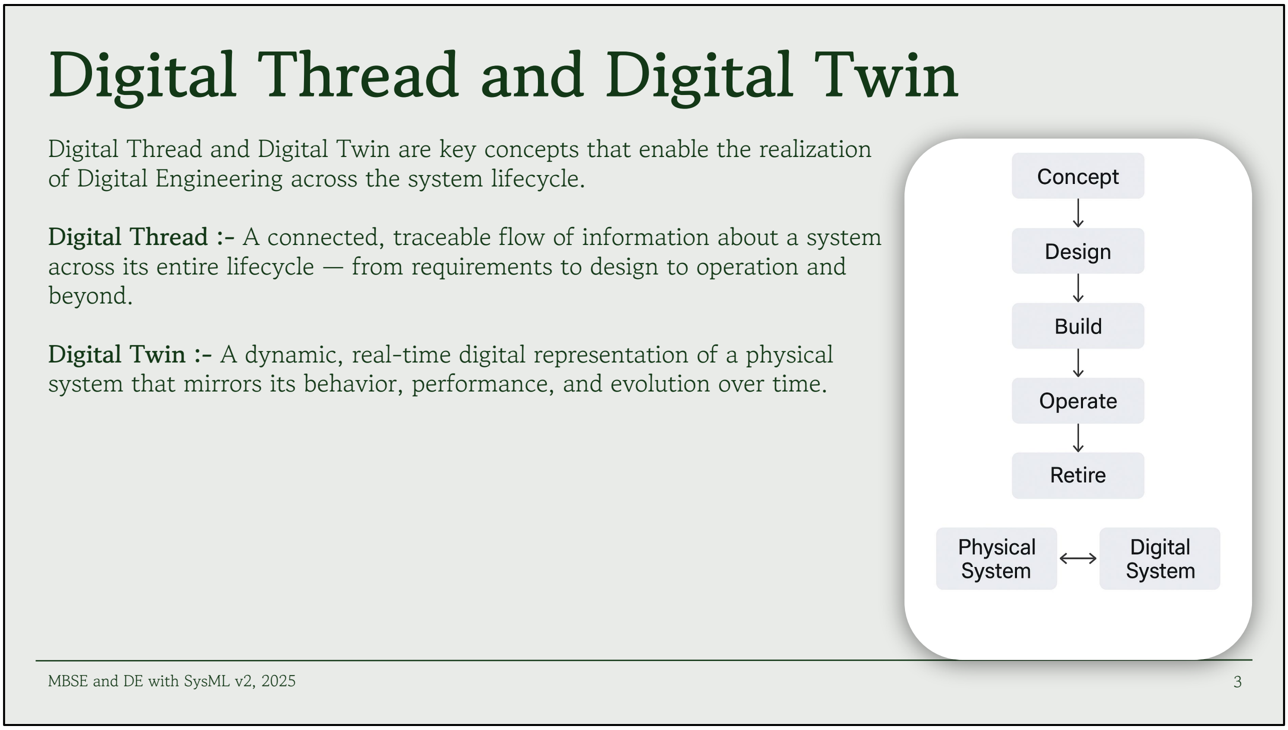
At the heart of Digital Engineering is the idea of the **Digital Thread** — a seamless flow of information that traces every decision, every requirement, and every test result across time.

And supporting the Digital Thread is the **Digital Twin** — a digital replica of the real-world system that can be analyzed, simulated, and monitored in real-time.

Digital Engineering:

- Improves **traceability** — so you can always connect design decisions back to stakeholder needs.
- Improves **collaboration** — because engineers, customers, and suppliers work from the same digital models.
- Enables **faster, smarter decisions** — by analyzing models instead of waiting for physical prototypes.
- Reduces **cost, risk, and time-to-market** — critical advantages in today’s competitive and high-risk industries.

Throughout this course, we will learn **how Systems Engineering fits into Digital Engineering**, and how techniques like **MBSE** and languages like **SysML v2** enable this digital future.



Digital Thread and Digital Twin

Digital Thread and Digital Twin are key concepts that enable the realization of Digital Engineering across the system lifecycle.

Digital Thread :- A connected, traceable flow of information about a system across its entire lifecycle — from requirements to design to operation and beyond.

Digital Twin :- A dynamic, real-time digital representation of a physical system that mirrors its behavior, performance, and evolution over time.

To fully understand Digital Engineering, it’s crucial to grasp two important, closely related concepts: **Digital Thread** and **Digital Twin**.

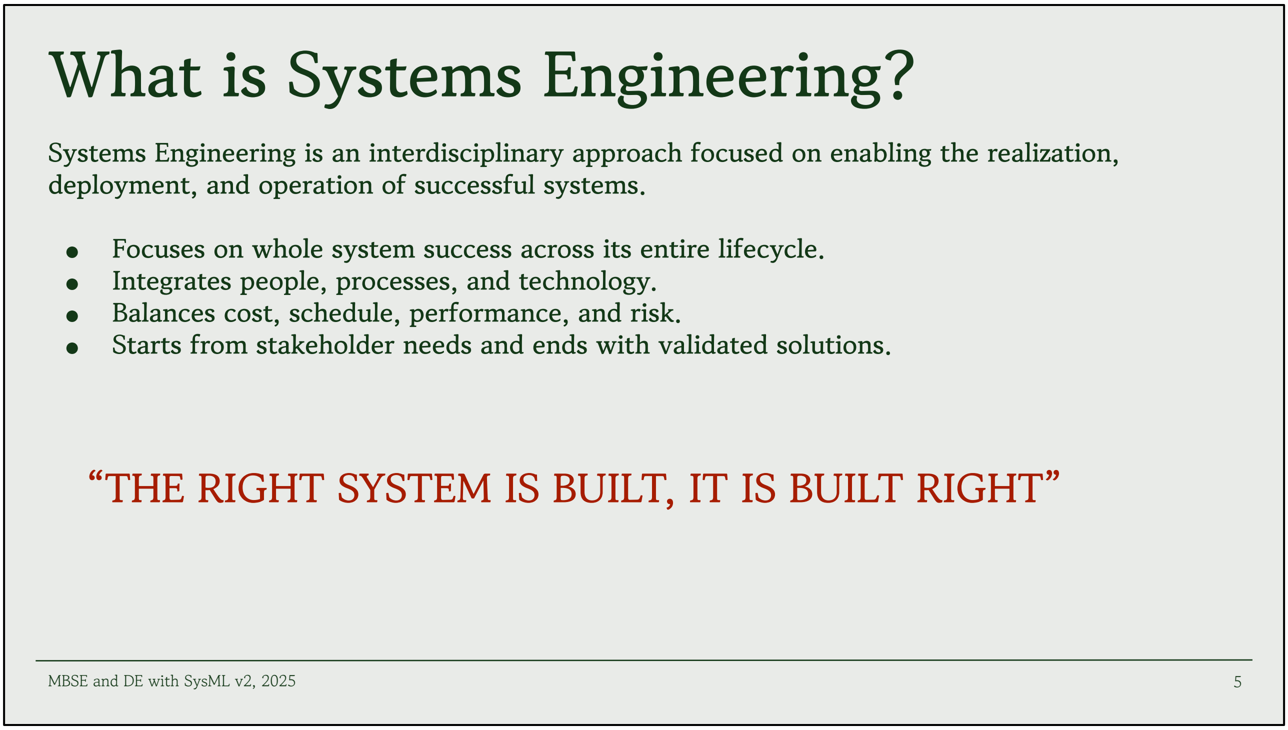
First, the **Digital Thread**: Imagine every decision, requirement, design change, test result, and operational feedback — all **linked together** in a seamless, traceable chain of information. That’s the Digital Thread. It connects all activities from the earliest system concept, through detailed design and production, into maintenance, and even into system disposal. The Digital Thread ensures that nothing is lost, and that you can always trace back and understand **why the system is the way it is**. It improves accountability, reduces rework, and allows better decision-making across the entire system lifecycle.

Second, the **Digital Twin**: The Digital Twin is a **dynamic digital replica** of the real-world physical system. It’s not just a 3D model — it’s a living, evolving system that mirrors the physical system’s behavior, performance, and health in real-time or near-real-time. The Digital Twin can be used to simulate operations, predict failures, optimize performance, and even support autonomous system decision-making.

Importantly, **Digital Thread and Digital Twin work together**:

- The Digital Thread captures the history and rationale behind the system.
- The Digital Twin represents the system’s real-time behavior and evolution.

Both of these are central to modern Digital Engineering — and later, you’ll see how Systems Engineering, MBSE, and SysML v2 contribute directly to making them possible.



What is Systems Engineering?

Systems Engineering is an interdisciplinary approach focused on enabling the realization, deployment, and operation of successful systems.

- Focuses on whole system success across its entire lifecycle.
- Integrates people, processes, and technology.
- Balances cost, schedule, performance, and risk.
- Starts from stakeholder needs and ends with validated solutions.

“THE RIGHT SYSTEM IS BUILT, IT IS BUILT RIGHT”

Now that we understand the big vision of Digital Engineering and the concepts of Digital Thread and Digital Twin, let’s focus on one of the most fundamental disciplines that makes this all possible: **Systems Engineering**.

Systems Engineering is the art and science of ensuring that complex systems — whether they are spacecraft, power grids, automobiles, or healthcare systems — are **designed, built, and operated successfully**.

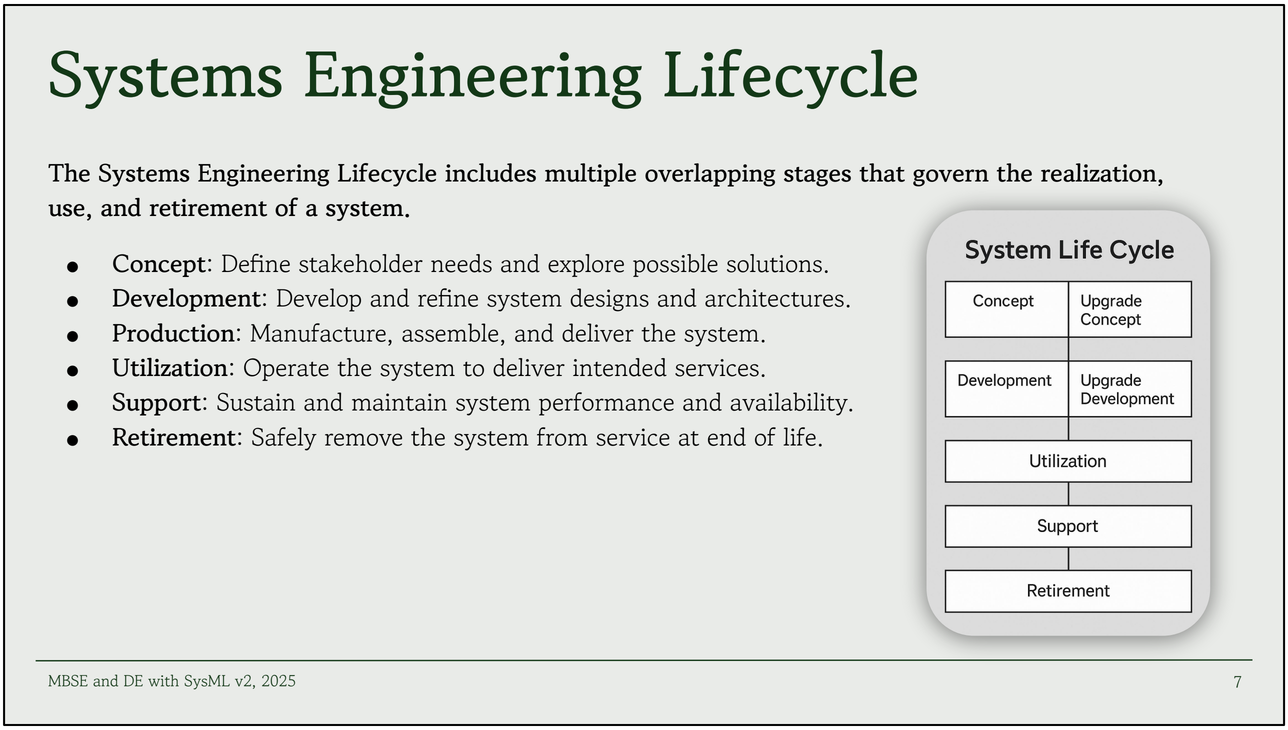
It is an **interdisciplinary approach** because it doesn’t just focus on one domain like mechanical, electrical, or software engineering. Instead, it **integrates all disciplines**, managing the relationships, interactions, and trade-offs between different subsystems.

The focus of Systems Engineering is always on the **whole system** — how all the parts work together to satisfy **stakeholder needs** — from the very beginning concept, through design, development, testing, deployment, operations, maintenance, and eventual retirement. At every stage, Systems Engineering carefully **balances competing factors**:

- **Cost**: Staying within budget.
- **Schedule**: Meeting deadlines and milestones.
- **Performance**: Achieving technical goals and mission success.
- **Risk**: Managing uncertainties to avoid failures.

And importantly, Systems Engineering begins with understanding **what stakeholders truly need**, even if they don't know how to express it clearly. The job of the Systems Engineer is to **translate vague desires into precise requirements**, design solutions that satisfy them, and prove through **verification and validation** that the system really works as intended.Without Systems Engineering, large, complex projects almost always fail — not because of technical challenges alone, but because of miscommunication, missed requirements, unexpected risks, and fragmented development efforts. In the world of Digital Engineering, Systems Engineering

becomes even more important — because it forms the foundation upon which the digital models, the Digital Thread, and the Digital Twin are built and maintained.
In short: **Systems Engineering ensures that the right system is built — and that it is built right.**



Systems Engineering Lifecycle

The Systems Engineering Lifecycle includes multiple overlapping stages that govern the realization, use, and retirement of a system.

- **Concept**: Define stakeholder needs and explore possible solutions.
- **Development**: Develop and refine system designs and architectures.
- **Production**: Manufacture, assemble, and deliver the system.
- **Utilization**: Operate the system to deliver intended services.
- **Support**: Sustain and maintain system performance and availability.
- **Retirement**: Safely remove the system from service at end of life.

System Life Cycle

Concept | Upgrade Concept

Development | Upgrade Development

Utilization

Support

Retirement

Now let's explore the **official Systems Engineering Lifecycle** as described in the INCOSE Systems Engineering Handbook. Unlike a simple sequence, the **System Life Cycle** involves multiple **overlapping and interacting stages**. Each stage represents a distinct phase in the life of a system — from the earliest ideas all the way to final retirement.

The first stage is the **Concept Stage**: Here we focus on **understanding stakeholder needs**, defining the system's goals, and exploring different solution options.It's during this phase that we decide **what the system should do** and **why**.

Next comes the **Development Stage**: This involves the **detailed design** and **architectural refinement** of the system. Here we model the system, allocate functionality to components, select technologies, and prepare for production.

Following development is the **Production Stage**: This is where the system is **manufactured**, **assembled**, and **delivered**. The transition from development to production must be carefully managed to ensure that the system is built according to specifications.

After production, we enter the **Utilization Stage**: This is when the system is actually **used** to deliver services and value to stakeholders. It includes monitoring system operations and gathering performance feedback.

Running parallel to utilization is the **Support Stage**: Support activities ensure that the system remains **available**, **reliable**, and **performing optimally** over time.

Support includes maintenance, upgrades, repairs, and service logistics.

Finally, every system eventually enters the **Retirement Stage**: When the system has reached the end of its useful life — whether due to obsolescence, performance degradation, or new technologies — it must be **retired safely and responsibly**.

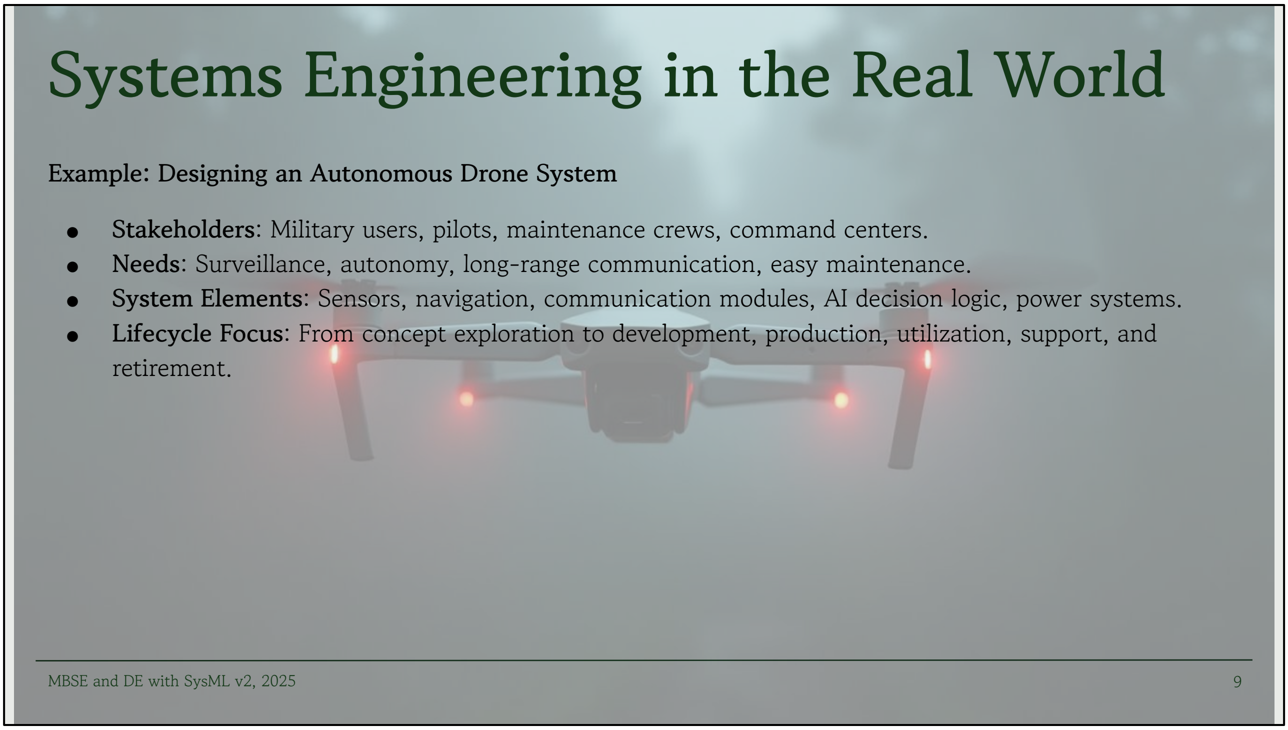
Notice also that the INCOSE lifecycle model includes **upgrades** —

- Upgrade Concept

- Upgrade Development
- Upgrade Production

These recognize that **many systems are enhanced over time** to extend their useful life, adapt to new requirements, or take advantage of new technologies.

Understanding this lifecycle model is fundamental to Systems Engineering practice: It helps engineers **plan activities**, **manage risks**, and **sustain system value** across years or even decades.



Systems Engineering in the Real World

Example: Designing an Autonomous Drone System

- **Stakeholders**: Military users, pilots, maintenance crews, command centers.
- **Needs**: Surveillance, autonomy, long-range communication, easy maintenance.
- **System Elements**: Sensors, navigation, communication modules, AI decision logic, power systems.
- **Lifecycle Focus**: From concept exploration to development, production, utilization, support, and retirement.

Now that we understand the official Systems Engineering Lifecycle, let’s bring it to life with a practical real-world example: **Designing an Autonomous Drone System**.

First, we start — as always — by identifying the **stakeholders**:

- The **military users** who depend on drone missions for surveillance.
- The **pilots and operators** who must interact with the drone systems easily and reliably.
- The **maintenance crews** who must diagnose, repair, and upgrade drones under field conditions.
- The **command centers** that depend on real-time communication, data gathering, and mission control.

From these stakeholders, we define **needs**:

- The drone must perform **surveillance** in contested environments.
- It must maintain **long-range communication**.
- It must operate with **high autonomy** — making decisions without human input if communication is disrupted.
- It must be **easy to maintain** to minimize downtime during missions.

Based on these needs, Systems Engineers develop the **system elements**:

- **Sensors** for imaging, positioning, and environmental awareness.
- **Navigation Systems** using GPS, inertial measurement units (IMUs), and visual navigation.
- **Communication Modules** supporting secure, long-range, and redundant links.
- **AI Decision-Making Modules** allowing the drone to interpret situations and act autonomously.
- **Power Systems** — such as battery packs — sized for mission endurance and optimized for easy replacement.

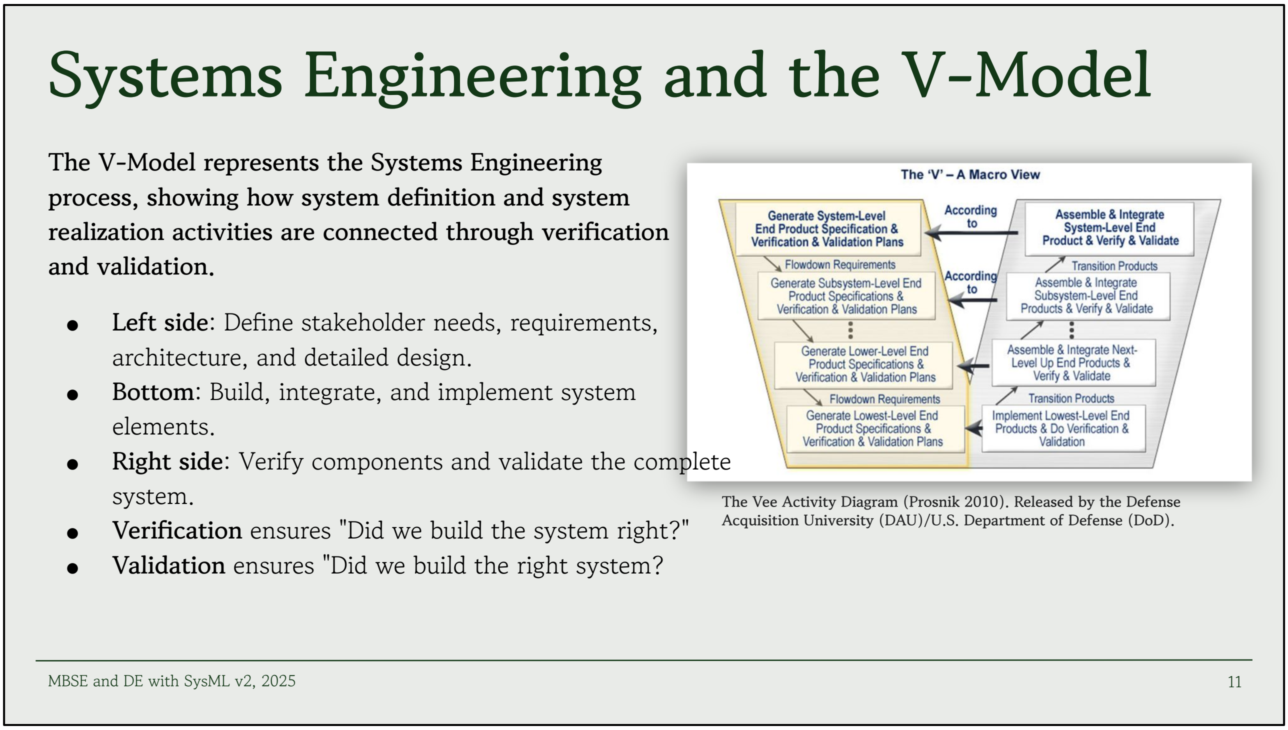
Applying the Systems Engineering Lifecycle:

- During the **Concept Stage**, alternative drone architectures and mission profiles would be evaluated.
- In **Development**, detailed models of drone structure, behavior, and communications would be created and validated.
- In **Production**, manufacturing processes and quality assurance systems would be set up.
- During **Utilization**, flight operations would be monitored, data collected, and missions supported.
- **Support** teams would maintain the drones, handle repairs, and implement system upgrades over time.
- Finally, at **Retirement**, drones would be decommissioned responsibly, managing sensitive technologies and materials.

This simple example shows that Systems Engineering is **not just about designing the drone** — It's about **managing the entire lifecycle**, **for all stakeholders**, **over the full system lifespan**.

Without Systems Engineering, projects like this would suffer from missed needs, integration failures, costly rework, and operational surprises.

In Digital Engineering, these lifecycle activities are increasingly modeled, analyzed, and connected through the Digital Thread and Digital Twin concepts we discussed earlier — and that's what makes modern Systems Engineering even more powerful.



Systems Engineering and the V-Model

The V-Model represents the Systems Engineering process, showing how system definition and system realization activities are connected through verification and validation.

- **Left side**: Define stakeholder needs, requirements, architecture, and detailed design.
- **Bottom**: Build, integrate, and implement system elements.
- **Right side**: Verify components and validate the complete system.
- **Verification** ensures "Did we build the system right?"
- **Validation** ensures "Did we build the right system?

The 'V' – A Macro View
Generate System-Level End Product Specification & Verification & Validation Plans
According to
Assemble & Integrate System-Level End Product & Verify & Validate
Flowdown Requirements
Transition Products
Generate Subsystem-Level End Product Specifications & Verification & Validation Plans
According to
Assemble & Integrate Subsystem-Level End Products & Verify & Validate
Generate Lower-Level End Product Specifications & Verification & Validation Plans
Assemble & Integrate Next-Level Up End Products & Verify & Validate
Flowdown Requirements
Transition Products
Generate Lowest-Level End Product Specifications & Verification & Validation Plans
Implement Lowest-Level End Products & Do Verification & Validation

The Vee Activity Diagram (Prosnik 2010). Released by the Defense Acquisition University (DAU)/U.S. Department of Defense (DoD).

To further understand how Systems Engineering organizes activities across the lifecycle, we introduce one of the most important models in engineering practice: **The V-Model**. The V-Model is a visual representation of the relationship between **system definition** activities and **system realization** activities.
Let's walk through it:

- On the **left side** of the 'V',we focus on **defining the system**:
  - First, **understanding stakeholder needs** and translating them into formal **requirements**.
  - Then, **developing the system architecture** — defining how the system is organized into components and how these components interact.
  - Finally, **detailing the design** — specifying the exact behaviors, interfaces, and constraints of each component.
- At the **bottom point** of the V, we move into **implementation and integration**:
  - Here, the physical or software components are **built**, **coded**, **manufactured**, or **procured**.
  - Then they are **integrated together** into larger subsystems or the full system.
- On the **right side** of the V, we focus on **verification and validation**:
  - We **verify** at each level that what we built matches the corresponding design specifications — 'Did we build the system right?'
  - And we **validate** that the complete system, when assembled, fulfills the original stakeholder needs — 'Did we build the right system?'

Each definition step on the left has a corresponding verification step on the right. For example:

- High-level system requirements are validated by operational tests.

- Subsystem designs are verified through subsystem integration tests.
- Component designs are verified through unit testing.

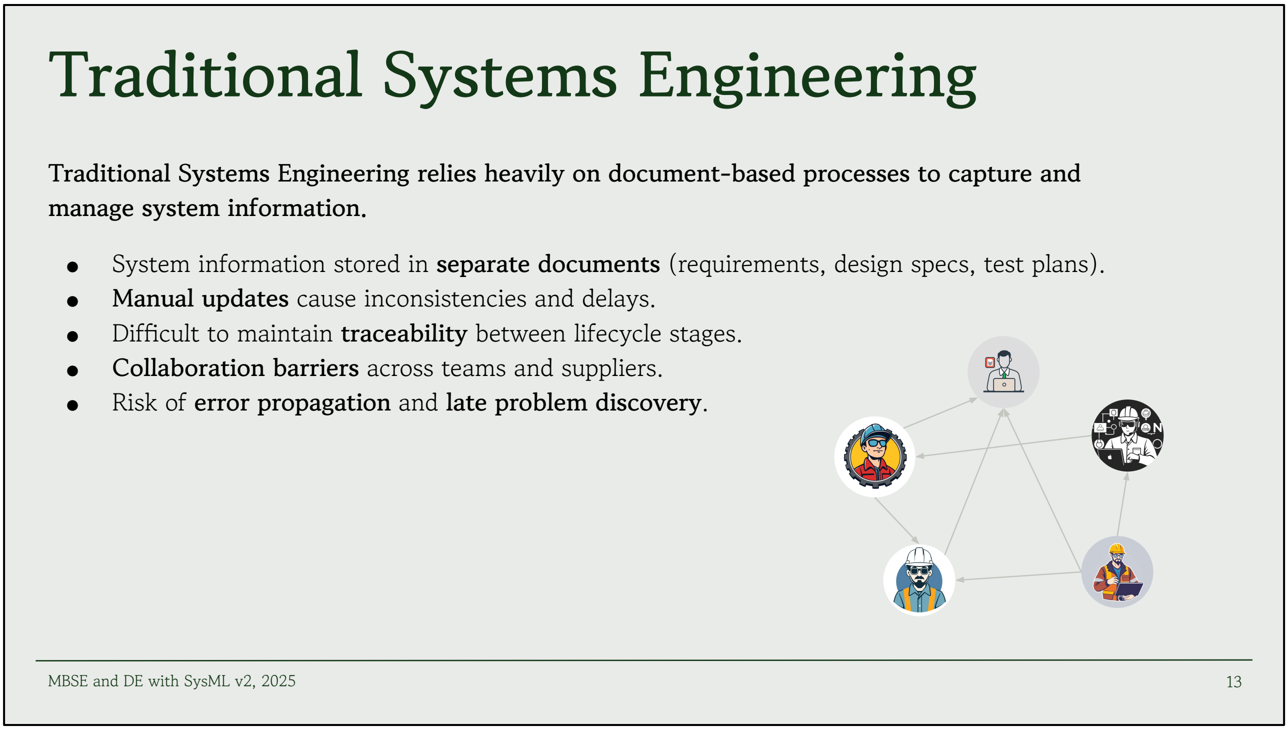
The V-Model emphasizes that **early activities drive later outcomes**:

- Good requirements lead to good tests.
- Clear architectures lead to effective integration.
- Solid verification strategies avoid costly surprises at the end.

It also emphasizes that **verification and validation are not afterthoughts** — they must be planned starting at the very beginning.

In modern Digital Engineering practice, models created during the left side of the V — using languages like SysML v2 — can be **analyzed, simulated, and even verified automatically** to support faster and more reliable development.

In short: **The V-Model helps Systems Engineers manage complexity and assure quality, from concept to reality.**



Traditional Systems Engineering

Traditional Systems Engineering relies heavily on document-based processes to capture and manage system information.

- System information stored in **separate documents** (requirements, design specs, test plans).
- **Manual updates** cause inconsistencies and delays.
- Difficult to maintain **traceability** between lifecycle stages.
- **Collaboration barriers** across teams and suppliers.
- Risk of **error propagation** and **late problem discovery**.

Before we discuss Model-Based Systems Engineering, it's important to understand how Systems Engineering has been practiced traditionally for decades. In **Traditional Systems Engineering**, system information — including requirements, architectures, designs, and test plans — is primarily managed through **documents**. Each major artifact exists as a **separate document**:

- A Word file for system requirements.
- A Visio or PowerPoint file for architecture diagrams.
- A separate Excel spreadsheet for test plans.
- Another document for interface definitions.

This document-centric approach creates multiple challenges:

First, there's a **manual update burden**: Whenever one document changes — say, a design spec — engineers must manually update related documents like the requirements baseline and test plans. It's easy to miss something, creating **inconsistencies** that may not be caught until late in the project.

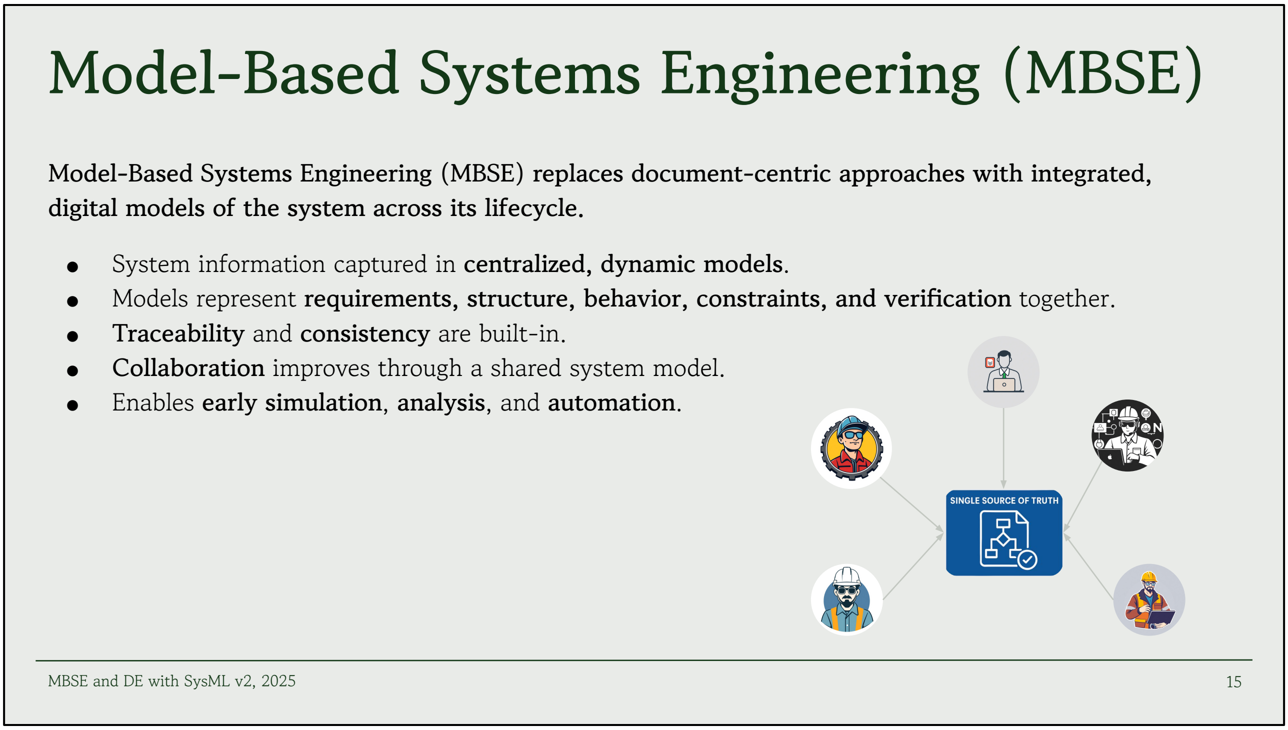
Second, **traceability is difficult**: Because information is spread across many files, it's hard to prove that every requirement has been implemented and verified. And when a requirement changes, identifying the impact across design, test, and validation artifacts is slow and error-prone.

Third, **collaboration becomes inefficient**: Different teams — electrical, mechanical, software, testing — often work in silos, passing documents back and forth by email or shared folders. There's no single, dynamic, shared view of the system.

Finally, **errors propagate silently**: Small inconsistencies in documents can lead to integration problems, performance failures, and even mission failure — and often these issues are only discovered late during testing, when fixes are expensive.

Traditional Systems Engineering worked reasonably well for small, relatively simple systems.

But for today’s complex, multi-disciplinary, fast-evolving systems, it’s simply too risky, too slow, and too costly That’s why the Systems Engineering community evolved toward a better way — one based not on disconnected documents, but on **integrated digital models**.
And that leads us directly to our next topic: **Model-Based Systems Engineering, or MBSE.**



Model-Based Systems Engineering (MBSE)

Model-Based Systems Engineering (MBSE) replaces document-centric approaches with integrated, digital models of the system across its lifecycle.

- System information captured in **centralized, dynamic models**.
- Models represent **requirements, structure, behavior, constraints, and verification** together.
- **Traceability** and **consistency** are built-in.
- **Collaboration** improves through a shared system model.
- Enables **early simulation**, **analysis**, and **automation**.

After understanding the challenges of Traditional Systems Engineering, we now introduce a major shift that is transforming the way systems are engineered: **Model-Based Systems Engineering, or MBSE**. MBSE fundamentally changes the way system information is managed. Instead of relying on disconnected documents, MBSE captures **requirements**, **designs**, **behaviors**, **constraints**, and **verification plans** in a **centralized, dynamic digital model**.

In MBSE, the **model is the authoritative source of information** — not the documents. This brings multiple powerful advantages:

First, **traceability and consistency are built-in**:

- Each requirement, component, function, and test case is linked inside the model.
- Changes to one part automatically highlight impacts on others.
- You can easily trace how a stakeholder need leads to a requirement, leads to a design decision, leads to a verification activity.

Second, **collaboration dramatically improves**:

- Different engineering disciplines — mechanical, electrical, software, test — all interact with the same model.
- Everyone shares a **common, current understanding** of the system at all times.

Third, MBSE enables **early simulation, analysis, and verification**:

- Instead of waiting for physical prototypes, engineers can simulate behavior, run performance analyses, and verify design compliance at the model level.
- This leads to earlier detection of problems, faster iteration cycles, and major cost savings.

Fourth, MBSE enables **automation**:

- Models can be automatically checked for completeness, consistency, and compliance.
- They can feed into downstream tools for detailed design, testing, and even operations

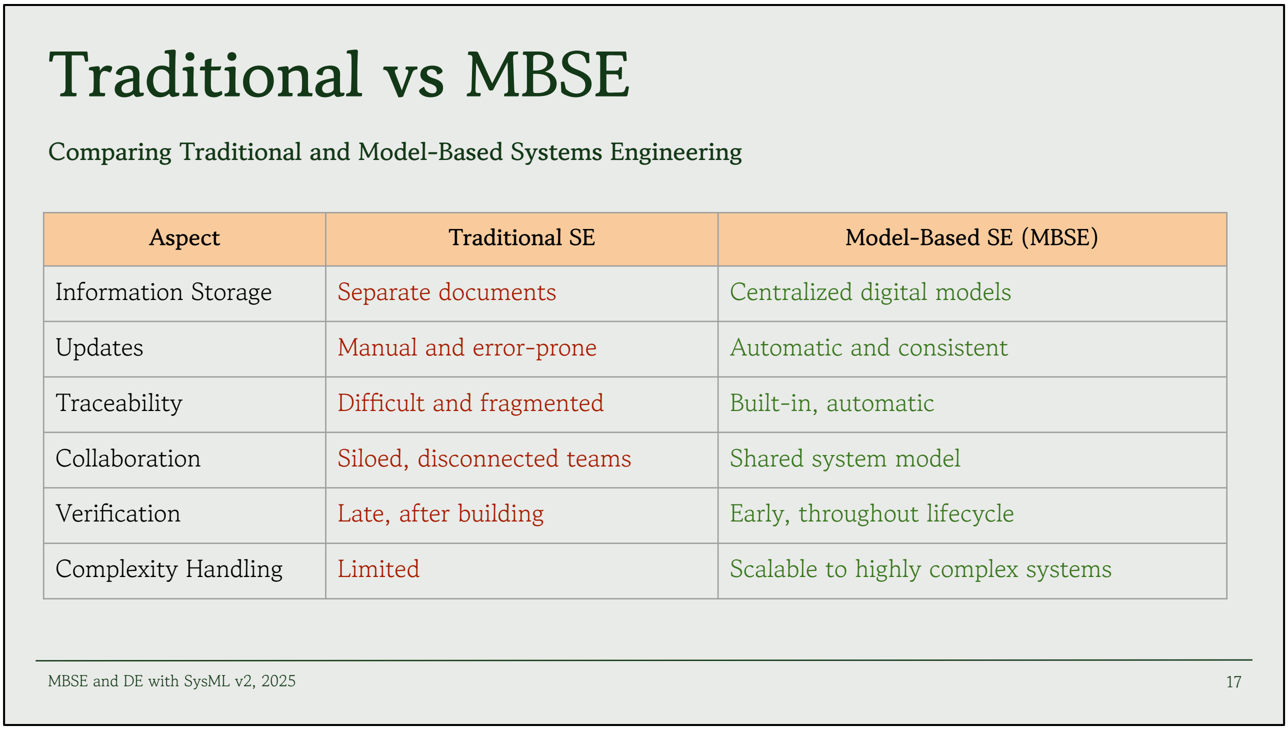
- monitoring.

MBSE is a natural fit for the demands of modern Digital Engineering:

- It supports the **Digital Thread**, enabling seamless flow of information across the lifecycle.
- It supports the creation of **Digital Twins**, because the real-world system behavior is anchored in the authoritative system model.

Importantly, MBSE is not just about 'drawing diagrams' — It’s about **formalizing system knowledge** in a precise, analyzable, and actionable way.

In short: **MBSE enables engineers to design, verify, and manage systems more effectively, more reliably, and more collaboratively — across the entire system lifecycle.**



Traditional vs MBSE

Comparing Traditional and Model-Based Systems Engineering

Aspect	Traditional SE	Model-Based SE (MBSE)
Information Storage	Separate documents	Centralized digital models
Updates	Manual and error-prone	Automatic and consistent
Traceability	Difficult and fragmented	Built-in, automatic
Collaboration	Siloed, disconnected teams	Shared system model
Verification	Late, after building	Early, throughout lifecycle
Complexity Handling	Limited	Scalable to highly complex systems

Now that we’ve discussed Traditional Systems Engineering and MBSE separately, let’s compare them side-by-side to see the difference clearly.

First, on **information storage**: Traditional SE stores information in separate documents — requirements in one file, architecture in another, test plans in yet another. MBSE centralizes this information into an integrated digital model that captures requirements, design, behavior, constraints, and verification in one place.

Second, on **updates**: In Traditional SE, updates are manual and error-prone. In MBSE, updates are automated within the model, reducing errors and ensuring that changes ripple appropriately across related elements.

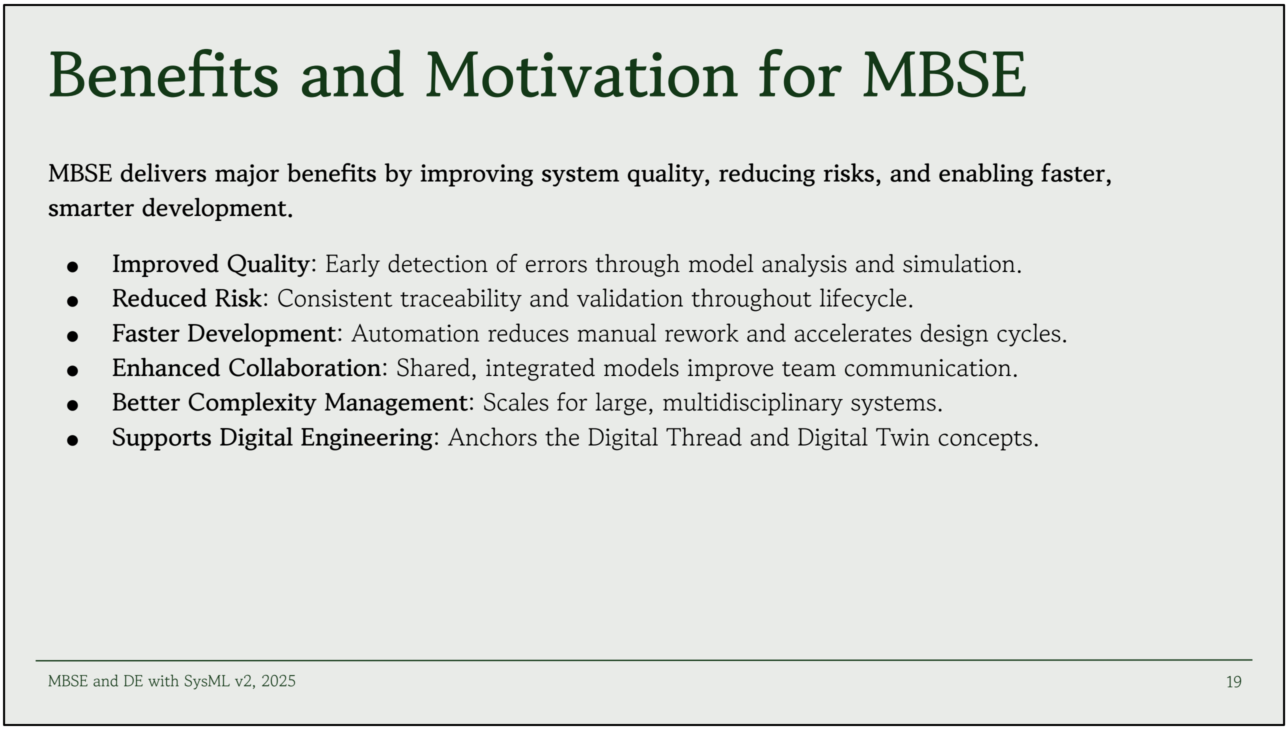
Third, on **traceability**: In Traditional SE, traceability between needs, requirements, design elements, and tests must be manually maintained — often inconsistently. In MBSE, traceability is built into the model structure and can be automatically analyzed and visualized.

Fourth, on **collaboration**: Traditional SE tends to isolate disciplines — mechanical, electrical, software, testing teams each work in their own documents. MBSE promotes collaboration because all disciplines access and interact with the same integrated model.

Fifth, on **verification**: In Traditional SE, verification activities are often delayed until after physical prototypes are built — leading to costly late-stage problem discovery. In MBSE, verification can happen early and continuously by simulating models, checking consistency, and running early validations.

Finally, on **handling system complexity**: Traditional document-centric methods struggle with the increasing size and complexity of modern systems. MBSE, by using formal models and automation, scales to handle the vast complexity of today’s multi-domain, software-intensive systems.

In short: **MBSE is not just a modernization — it is a strategic necessity for the scale, speed,** and reliability demanded by today’s engineering challenges.



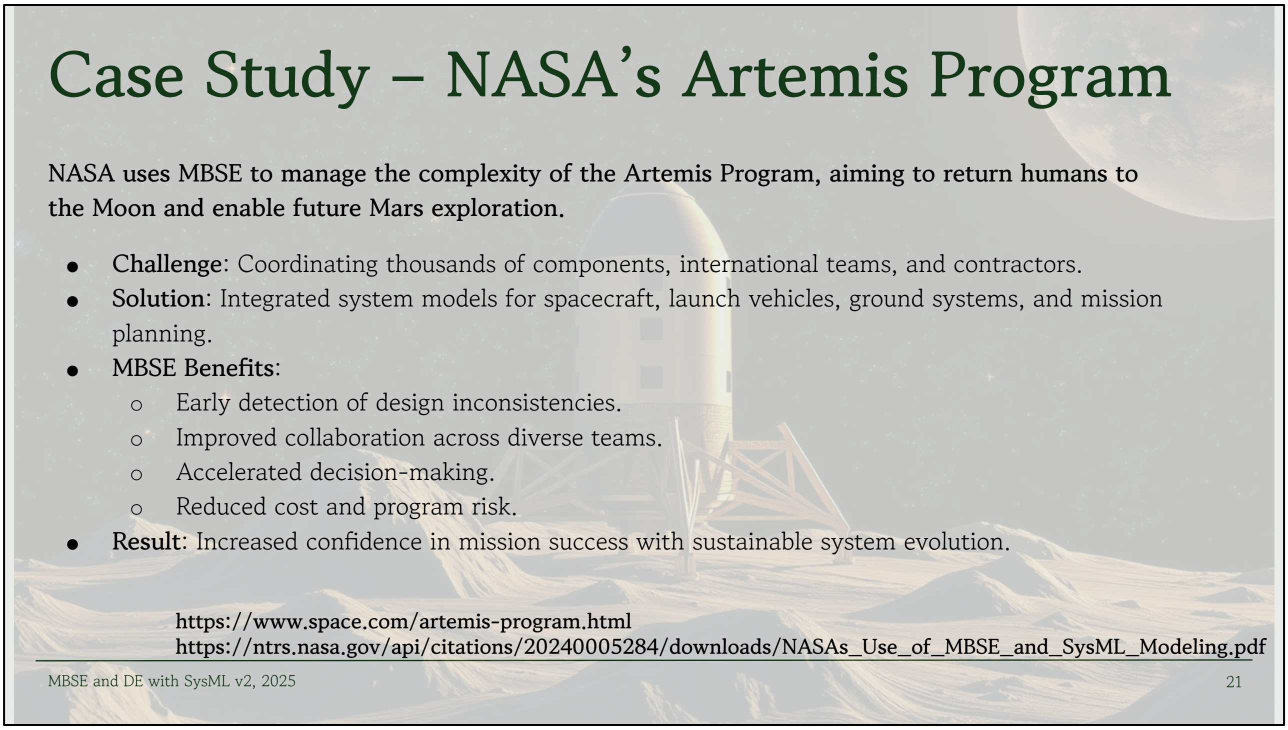
Benefits and Motivation for MBSE

MBSE delivers major benefits by improving system quality, reducing risks, and enabling faster, smarter development.

- **Improved Quality**: Early detection of errors through model analysis and simulation.
- **Reduced Risk**: Consistent traceability and validation throughout lifecycle.
- **Faster Development**: Automation reduces manual rework and accelerates design cycles.
- **Enhanced Collaboration**: Shared, integrated models improve team communication.
- **Better Complexity Management**: Scales for large, multidisciplinary systems.
- **Supports Digital Engineering**: Anchors the Digital Thread and Digital Twin concepts.

Now that we've compared Traditional Systems Engineering and MBSE, let's dive deeper into **why MBSE is not just useful — but essential** for modern systems engineering.
First, MBSE leads to **improved system quality**: Because we can simulate, analyze, and verify systems early in the development lifecycle, Mistakes and mismatches can be detected **before** physical prototypes are built — when they are cheaper and easier to fix.
Second, MBSE leads to **reduced project risk**: With built-in **traceability** and **consistency** across requirements, designs, behaviors, and tests, MBSE ensures that when changes occur, the impact is automatically tracked and understood — reducing the chances of late-stage surprises.
Third, MBSE enables **faster development cycles**: With formal models and automation, engineers can generate designs, check compliance, and run simulations automatically. This dramatically cuts down the time lost to manual document editing and rework.
Fourth, MBSE enhances **collaboration** across teams: Mechanical, electrical, software, test, and operations teams all access and contribute to the same integrated system model. Everyone stays aligned, reducing misunderstandings and costly handoff errors.
Fifth, MBSE allows better **management of system complexity**: Today's systems — autonomous vehicles, aerospace platforms, smart grids — involve massive numbers of interacting components and technologies. MBSE provides the structure, abstraction, and analysis tools needed to **scale up** without getting lost in the details.
Finally, MBSE **supports Digital Engineering directly**: MBSE models form the backbone of the **Digital Thread** — ensuring data and decisions flow seamlessly across the lifecycle. MBSE models feed into **Digital Twin** creation — allowing real-time monitoring and prediction based on the system model.
In short: **MBSE is the only practical way to engineer the complex, integrated, rapidly evolving systems that define the 21st century.**

It’s not just a nice-to-have — for many industries, it is becoming a critical strategic advantage.



Case Study – NASA’s Artemis Program

NASA uses MBSE to manage the complexity of the Artemis Program, aiming to return humans to the Moon and enable future Mars exploration.

- **Challenge**: Coordinating thousands of components, international teams, and contractors.
- **Solution**: Integrated system models for spacecraft, launch vehicles, ground systems, and mission planning.
- **MBSE Benefits**:
  - Early detection of design inconsistencies.
  - Improved collaboration across diverse teams.
  - Accelerated decision-making.
  - Reduced cost and program risk.
- **Result**: Increased confidence in mission success with sustainable system evolution.

https://www.space.com/artemis-program.html
https://ntrs.nasa.gov/api/citations/20240005284/downloads/NASAs_Use_of_MBSE_and_SysML_Modeling.pdf

Let’s now move from concepts to reality — and look at a true large-scale application of MBSE: **NASA’s Artemis Program**.

The **Artemis Program** is NASA’s ambitious effort to return humans to the Moon, establish sustainable lunar operations, and eventually push human exploration toward Mars.

The complexity of Artemis is staggering:

- It involves **spacecraft**, **launch vehicles**, **surface habitats**, **rover systems**, and **ground operations**.
- It requires coordination among **NASA centers**, **international space agencies**, and **private industry contractors**.
- Each system element has to integrate perfectly with others, often built by entirely different organizations, over long timeframes.

In the past, such a program would have generated thousands of independent documents — requirements specs, design diagrams, test plans — all maintained separately, often inconsistently. Instead, NASA adopted **Model-Based Systems Engineering** to manage Artemis.

They built **integrated system models** capturing:

- Requirements for each subsystem and the mission as a whole.
- Architectural designs showing how modules, components, and functions interact.
- Behavioral models simulating mission scenarios.
- Interfaces between spacecraft, ground systems, and human operators.

The MBSE approach provided key advantages:

- **Early detection of design inconsistencies** — engineers could catch mismatches and interface problems before hardware was built.
- **Improved collaboration** — diverse teams and partners worked from a common model, reducing miscommunication and rework.

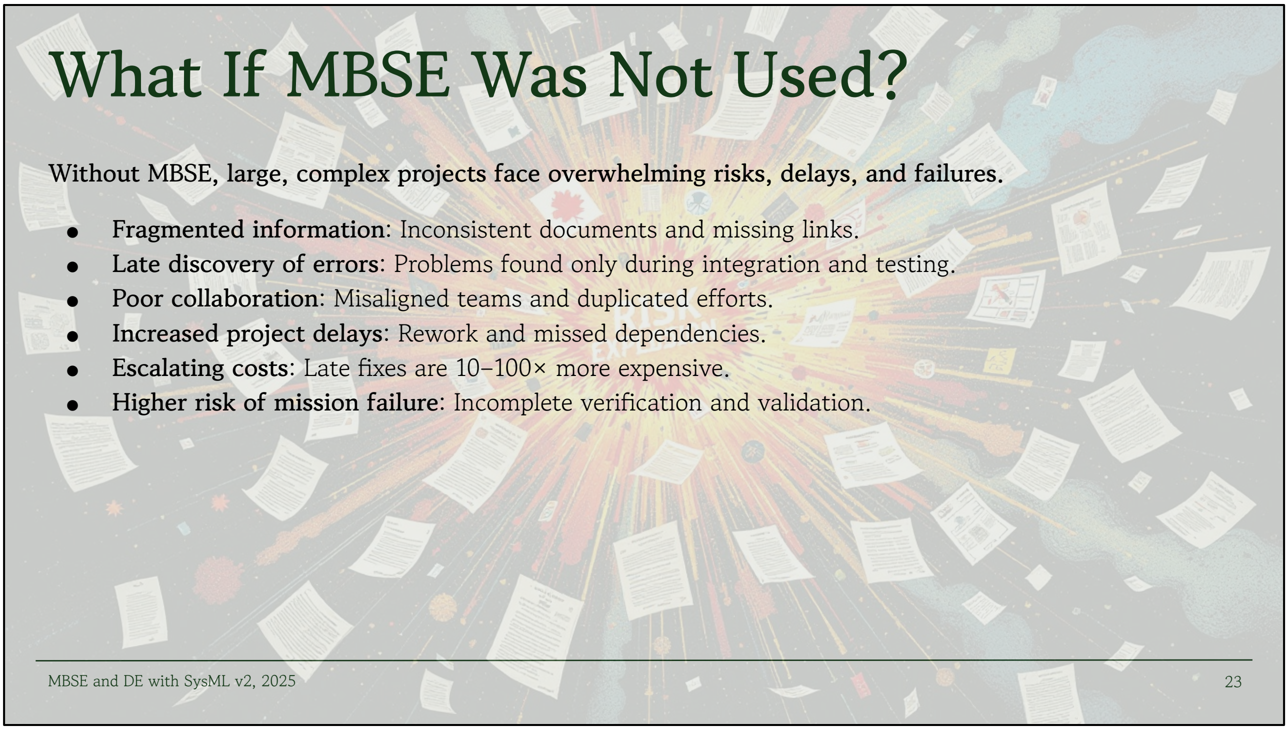
- **Faster decision-making** — leaders could analyze model simulations to choose between alternative designs with confidence.
- **Reduced cost and program risk** — by catching issues earlier and coordinating better across a vast project landscape.

Ultimately, MBSE gave NASA greater confidence that Artemis will succeed:

- Meeting mission needs.
- Staying within technical and budgetary constraints.
- Adapting to inevitable future changes, like adding new partners or evolving mission goals.

This case study shows that **MBSE is not theoretical — it is mission-critical.** If MBSE can help NASA manage the complexity of returning humans to the Moon — imagine what it can do for aerospace companies, energy providers, healthcare systems, and autonomous technologies.

MBSE is not just the future — it is happening now.



What If MBSE Was Not Used?

Without MBSE, large, complex projects face overwhelming risks, delays, and failures.

- **Fragmented information**: Inconsistent documents and missing links.
- **Late discovery of errors**: Problems found only during integration and testing.
- **Poor collaboration**: Misaligned teams and duplicated efforts.
- **Increased project delays**: Rework and missed dependencies.
- **Escalating costs**: Late fixes are 10–100× more expensive.
- **Higher risk of mission failure**: Incomplete verification and validation.

Let's take a moment to imagine: **What would happen if MBSE was not used** in large-scale, complex projects like NASA's Artemis Program, or major aerospace, automotive, and defense systems?

First, we would face **fragmented information**:

- Requirements would live in one set of documents, designs in another, tests in yet another.
- Keeping these consistent would be nearly impossible over years of development.
- Information would be duplicated, outdated, or missing critical links.

Second, **errors would be discovered late**: Without early model-based simulation and validation, problems would only surface during integration and physical testing — when fixing them is incredibly expensive and disruptive.

Third, we would experience **poor collaboration**:

- Different engineering teams would work in silos.
- Misalignments would cause rework, misinterpretations, and integration clashes.
- Stakeholders would lose confidence in project visibility and progress.

Fourth, **project delays would skyrocket**:

- Change impact analysis would be manual and error-prone.
- Missed dependencies would cause cascading failures.
- Teams would spend months or even years fixing avoidable issues.

Fifth, **costs would escalate dramatically**:

- According to engineering studies, **the cost to fix an error increases by 10 to 100 times** the later it is discovered in the lifecycle.
- Early modeling saves millions by catching errors before hardware or code is finalized.

Finally, and most importantly, there would be a **higher risk of mission failure**:

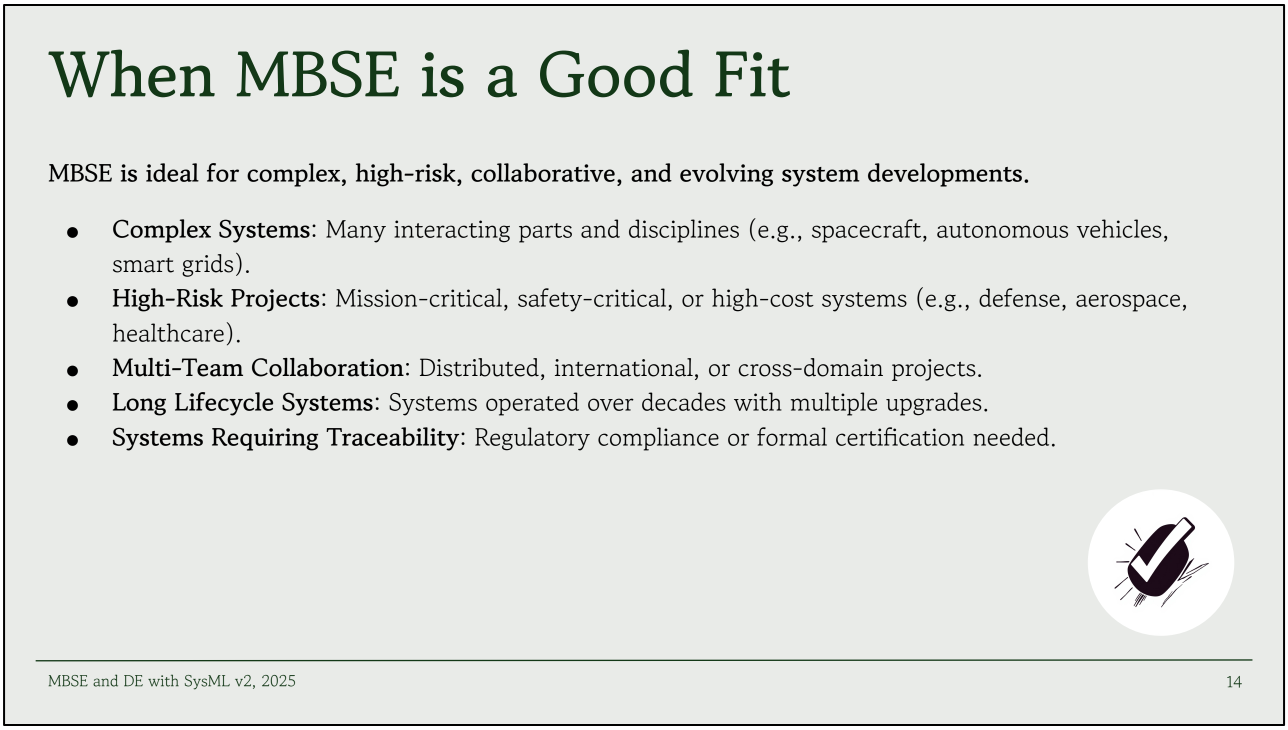
- Critical requirements might not be satisfied.
- Verification and validation gaps would go unnoticed.

Integration and operational failures could jeopardize the entire mission — with devastating financial, technical, and human consequences.

In short:Without MBSE,

- **Systems become fragile**.
- **Projects become slower and riskier**.
- **Innovation becomes harder**.

This is why today's leading industries — aerospace, defense, automotive, energy, healthcare — are investing heavily in MBSE: Because as systems become more complex, **modeling is not optional — it is essential.** MBSE gives us the ability to build complex, integrated systems with confidence, agility, and precision.



When MBSE is a Good Fit

MBSE is ideal for complex, high-risk, collaborative, and evolving system developments.

- **Complex Systems**: Many interacting parts and disciplines (e.g., spacecraft, autonomous vehicles, smart grids).
- **High-Risk Projects**: Mission-critical, safety-critical, or high-cost systems (e.g., defense, aerospace, healthcare).
- **Multi-Team Collaboration**: Distributed, international, or cross-domain projects.
- **Long Lifecycle Systems**: Systems operated over decades with multiple upgrades.
- **Systems Requiring Traceability**: Regulatory compliance or formal certification needed.

While MBSE provides strong advantages, it’s important to understand the **specific types of projects where MBSE truly shines**.

First, MBSE is ideal for **complex systems**:When a system involves many interacting parts, multiple domains (hardware, software, human operations), and evolving interfaces — traditional documents simply cannot keep up. MBSE's structured, integrated models can manage this complexity effectively.

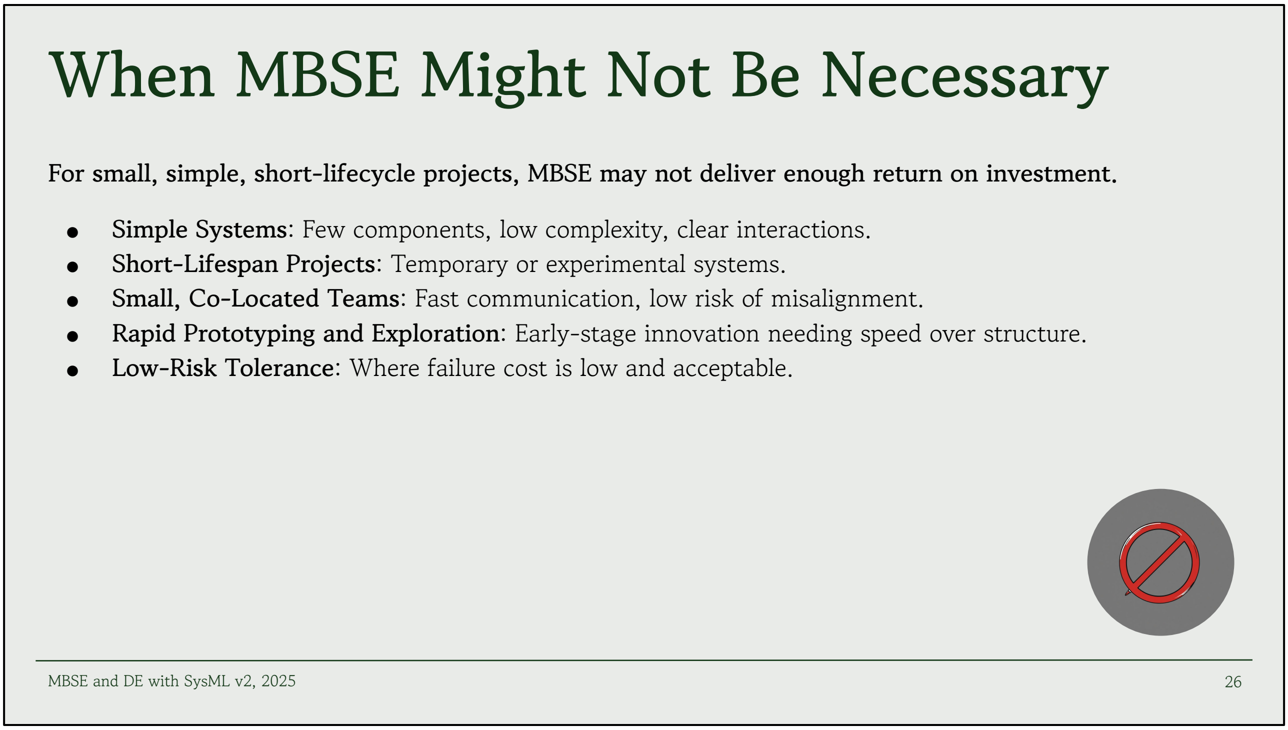
Second, MBSE is critical for **high-risk projects**: In industries like aerospace, defense, energy, and healthcare — where system failures can cause catastrophic loss of life, mission failure, or massive financial damage — Early detection of issues through modeling is invaluable.

Third, MBSE is a game-changer for **multi-team, collaborative projects**: When different teams — sometimes across different companies or countries — work together, MBSE provides a single, shared model that keeps everyone aligned on requirements, architecture, and interfaces.

Fourth, MBSE fits perfectly with **long lifecycle systems**: Systems that must operate for 10, 20, even 50 years — like satellites, submarines, transportation networks — Benefit tremendously from having a living, traceable, adaptable model across generations of upgrades.

Fifth, MBSE is essential when **regulatory compliance and traceability** are needed: For example, in aviation (FAA, EASA), automotive (ISO 26262), or medical devices (FDA), Authorities demand proof that every requirement has been satisfied and verified — MBSE supports this rigorously.

In short: **The bigger, riskier, more complex, and longer-lived the system — the more valuable MBSE becomes.** It’s not just about technology — it’s about managing complexity, ensuring success, and protecting lives and investments.



When MBSE Might Not Be Necessary

For small, simple, short-lifecycle projects, MBSE may not deliver enough return on investment.

- **Simple Systems**: Few components, low complexity, clear interactions.
- **Short-Lifespan Projects**: Temporary or experimental systems.
- **Small, Co-Located Teams**: Fast communication, low risk of misalignment.
- **Rapid Prototyping and Exploration**: Early-stage innovation needing speed over structure.
- **Low-Risk Tolerance**: Where failure cost is low and acceptable.

As much as MBSE offers tremendous benefits for complex, high-risk, long-term projects, it’s important to understand that **MBSE is not always the right tool for every project**.

There are cases where the **cost and effort to implement MBSE might outweigh the benefits**:

First, for **simple systems**: When a system has very few components, clear and limited interactions, and simple functionality, The overhead of building formal models might not be justified.

Second, for **short-lifespan projects**: Projects meant for temporary exhibitions, experiments, or one-off prototypes, May not benefit enough from formal traceability and lifecycle management.

Third, for **small, co-located teams**: If a handful of engineers are working together in the same room, Communication is fast and informal — maintaining a formal system model may not add significant value compared to lightweight documentation.

Fourth, during **rapid prototyping and exploration**: In the early phases of innovation — exploring ideas quickly, iterating designs daily — Speed and flexibility may be more important than structure and formal traceability. Here, informal models (sketches, quick diagrams) may be more appropriate initially.

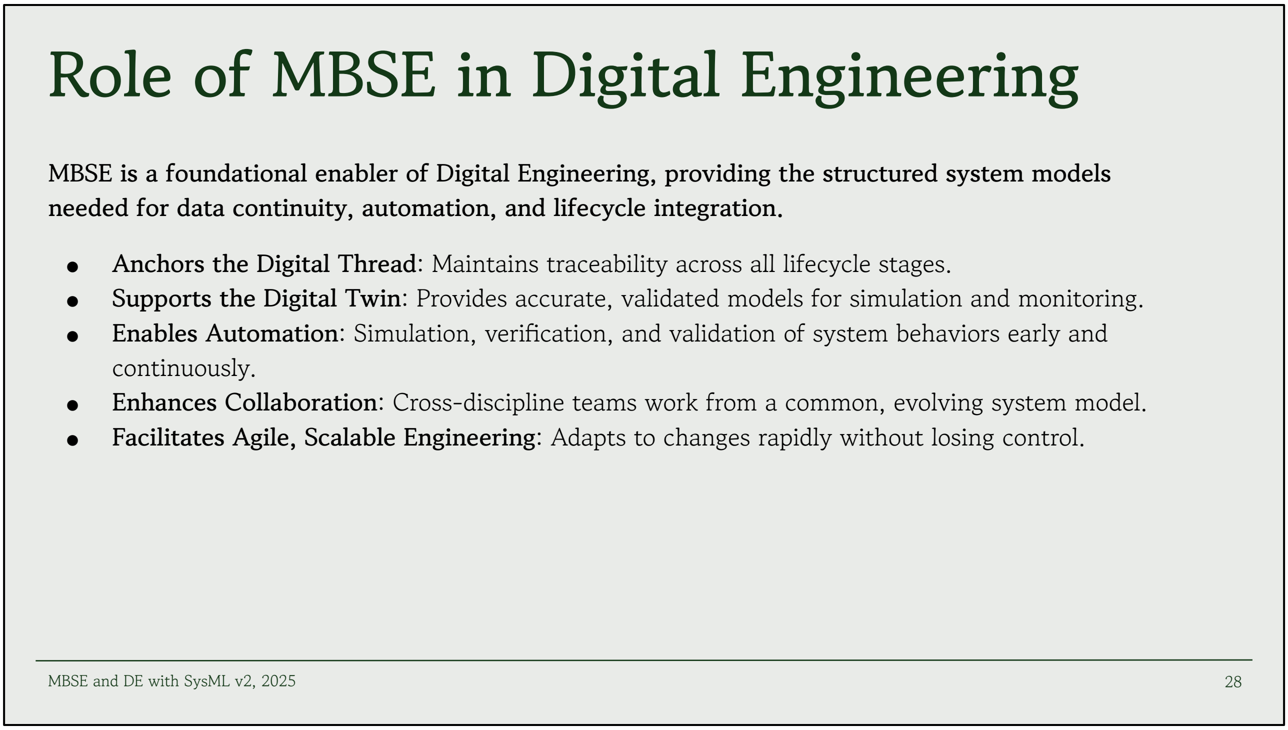
Fifth, when **the cost of failure is low and acceptable**: If a system failure causes minimal financial or operational impact, Organizations may choose to accept higher risk rather than invest in rigorous modeling.

However, it’s important to recognize:

- Even in these cases, as complexity or criticality grows, it may become necessary to transition into MBSE.
- MBSE is not all-or-nothing — it can be **introduced incrementally** as project complexity and risk evolve.

In short: **Good Systems Engineers make smart trade-offs** — They apply MBSE **where it brings**

real value, and use simpler methods where appropriate. MBSE **is a powerful tool — but like any tool, it must be used wisely.**



Role of MBSE in Digital Engineering

MBSE is a foundational enabler of Digital Engineering, providing the structured system models needed for data continuity, automation, and lifecycle integration.

- **Anchors the Digital Thread**: Maintains traceability across all lifecycle stages.
- **Supports the Digital Twin**: Provides accurate, validated models for simulation and monitoring.
- **Enables Automation**: Simulation, verification, and validation of system behaviors early and continuously.
- **Enhances Collaboration**: Cross-discipline teams work from a common, evolving system model.
- **Facilitates Agile, Scalable Engineering**: Adapts to changes rapidly without losing control.

After understanding the strengths and strategic application of MBSE, we now place it formally into the larger vision of **Digital Engineering**. **Digital Engineering** is not just about CAD models, simulations, or databases — It's about **using digital models to transform the entire engineering lifecycle**:

- From concept,
- To design,
- To production,
- To operation and support,
- All the way to system retirement.

At the **heart of Digital Engineering** is the need for a **structured, traceable, machine-readable system model** — and that's exactly where **Model-Based Systems Engineering** (MBSE) comes in.

First, MBSE **anchors the Digital Thread**:

- It connects stakeholder needs, requirements, design elements, behavior models, and verification plans across time.
- Every decision, change, and validation is captured inside a unified model — forming a continuous, navigable thread through the system's history.

Second, MBSE **supports the Digital Twin**:

- Accurate system models created through MBSE become the starting point for creating operational Digital Twins.
- These Digital Twins mirror real-world system behavior, enabling real-time monitoring, predictive maintenance, and performance optimization.

Third, MBSE **enables automation**:

- System behaviors can be simulated.

- Designs can be automatically checked against constraints and requirements.
- Verification and validation activities can begin **long before physical hardware is built** — reducing risk and cost dramatically.

Fourth, MBSE **enhances collaboration**: Engineers across different disciplines — electrical, mechanical, software, human factors — Work together on a common model, reducing miscommunication and accelerating decision-making.

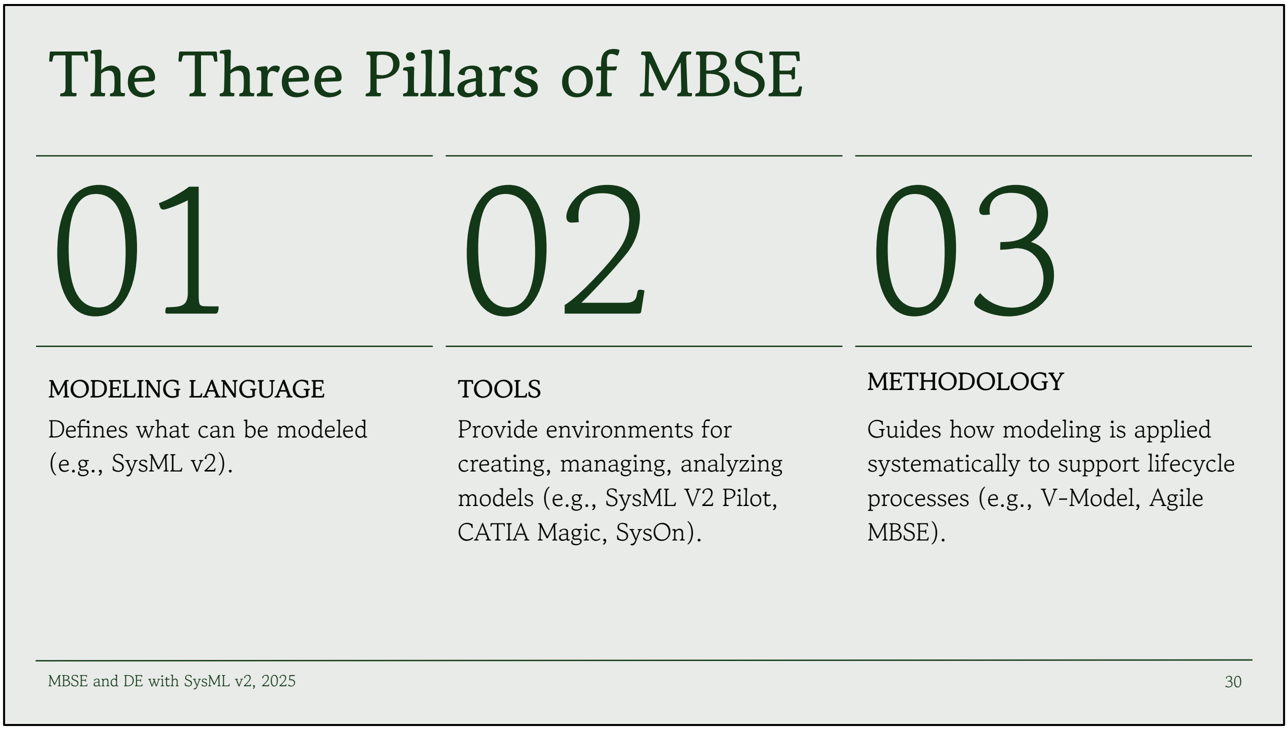
Fifth, MBSE **facilitates agile, scalable engineering**:

- As systems evolve, requirements change, and technology advances,
- MBSE models can be rapidly adapted without losing traceability or introducing chaos.

Without MBSE, Digital Engineering remains a fragmented dream.

With MBSE, Digital Engineering becomes an achievable, strategic transformation of how systems are engineered, maintained, and evolved.

In short: **MBSE is not just a tool inside Digital Engineering — it is the foundation upon which Digital Engineering is built.**



The Three Pillars of MBSE

01

MODELING LANGUAGE

Defines what can be modeled (e.g., SysML v2).

02

TOOLS

Provide environments for creating, managing, analyzing models (e.g., SysML V2 Pilot, CATIA Magic, SysOn).

03

METHODOLOGY

Guides how modeling is applied systematically to support lifecycle processes (e.g., V-Model, Agile MBSE).

Now that we understand the strategic role of MBSE in Digital Engineering, it's time to dive deeper into **how MBSE is actually made possible** in practice.

A successful MBSE approach stands firmly on **three essential pillars**:

First, the **Modeling Language**:

- The modeling language defines **what you can express and how you express it** within the system model.
- It provides the **grammar and vocabulary** that engineers use to define structure, behavior, requirements, constraints, and verification elements.
- Examples include languages like **SysML v2**, which we will be focusing on in this course.

Second, the **Tools**:

- Tools provide the **software environments** where engineers **create**, **manage**, **simulate**, and **analyze** models.
- A good tool enforces the rules of the modeling language and adds features like model validation, simulation integration, and version control.
- Examples of MBSE tools include **SYSMLV2 Pilot Implementation**, **CATIA Magic**, **SysOn** and others.
- Tools make the modeling language **usable at scale** and allow it to integrate into larger engineering ecosystems.

Third, the **Methodology**:

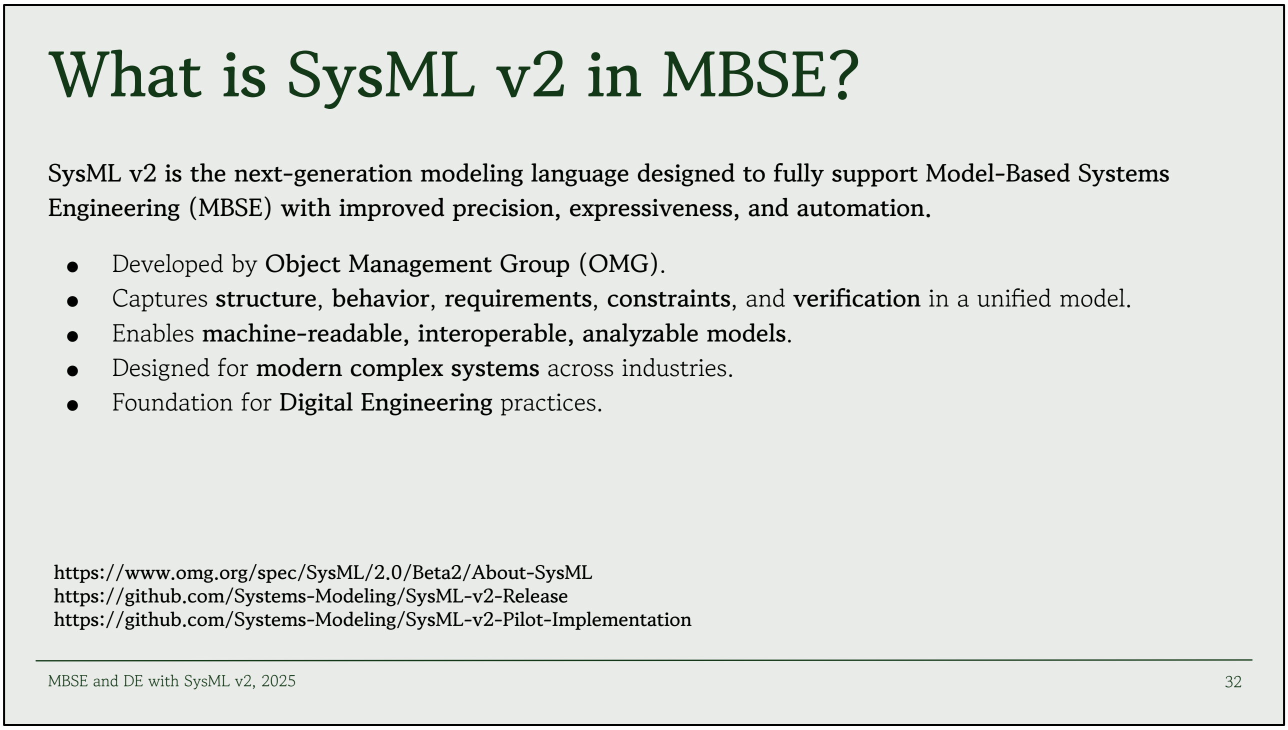
- Methodology defines **how the modeling work is organized and executed**.
- It connects the act of modeling to the **systems engineering lifecycle** — ensuring that modeling supports concept development, requirements engineering, architecture definition, integration, verification, validation, and evolution.

- Examples include the **V-Model approach**, **Agile MBSE frameworks**, or **Domain-Specific MBSE practices**.

Each pillar is **necessary but not sufficient alone**:

- A good language without good tools is painful to apply.
- Great tools without a clear methodology create disorganized models.
- A strong methodology without a powerful language and tools cannot deliver needed system precision.

In summary: **Effective MBSE is the balanced combination of Language, Tools, and Methodology — each reinforcing the others.** When these three pillars are strong and coordinated, MBSE becomes not just manageable, but transformative.



What is SysML v2 in MBSE?

SysML v2 is the next-generation modeling language designed to fully support Model-Based Systems Engineering (MBSE) with improved precision, expressiveness, and automation.

- Developed by **Object Management Group** (OMG).
- Captures **structure**, **behavior**, **requirements**, **constraints**, and **verification** in a unified model.
- Enables **machine-readable, interoperable, analyzable models**.
- Designed for **modern complex systems** across industries.
- Foundation for **Digital Engineering** practices.

https://www.omg.org/spec/SysML/2.0/Beta2/About-SysML
https://github.com/Systems-Modeling/SysML-v2-Release
https://github.com/Systems-Modeling/SysML-v2-Pilot-Implementation

Now that we understand that the first pillar of MBSE is the **Modeling Language**, let's talk specifically about **SysML v2** — the modern language designed to fully empower MBSE.

SysML stands for **Systems Modeling Language**. SysML v2 is the **next-generation standard**, developed by the **Object Management Group (OMG)**, after years of lessons learned from SysML v1 and the growing needs of complex system development. SysML v2 is built to address the increasing demands of **modern systems engineering** — systems that are larger, faster, more interconnected, and more software-intensive than ever before.

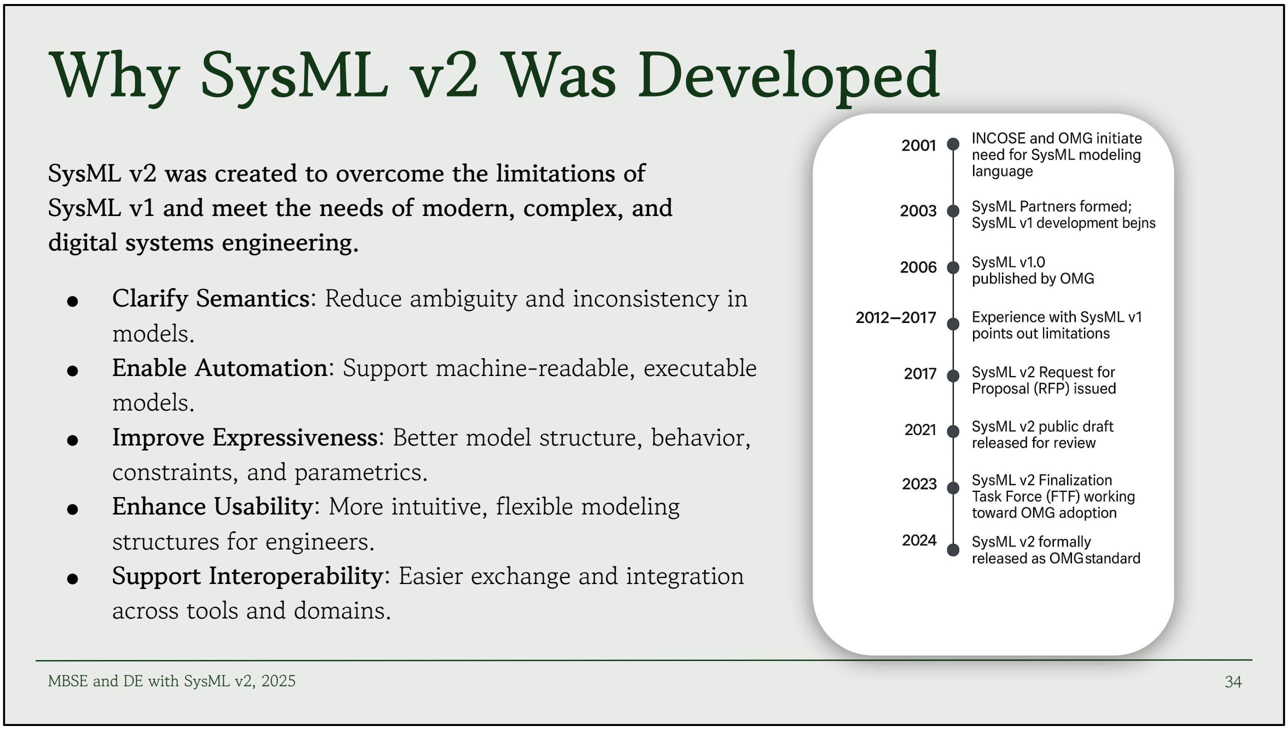
SysML v2 provides a **formal way to model** all key aspects of a system:

- **Structure**: How the system is organized — parts, components, connections, and hierarchies.
- **Behavior**: How the system acts, reacts, and operates over time — actions, interactions, scenarios, and states.
- **Requirements**: What the system must achieve — goals, needs, constraints, and performance targets.
- **Constraints**: Mathematical and logical conditions that must be satisfied — critical for engineering trade studies.
- **Verification and Validation**: How you prove that the system meets its requirements.

And importantly, SysML v2 is not just about creating nice diagrams. It is designed for:

- **Machine-readability**: Models can be analyzed, simulated, validated, and exchanged between tools without ambiguity.
- **Interoperability**: Different teams and tools can work with the same model structure consistently.
- **Automation**: Tasks like model checking, verification, and even partial code generation can be automated from the system model.

In the world of Digital Engineering, SysML v2 becomes the **core modeling language** that enables the Digital Thread, supports Digital Twin creation, and powers lifecycle data continuity.
SysML v2 is the language we will focus on throughout this course — Learning it will give you a professional foundation to participate in the future of systems engineering.



Why SysML v2 Was Developed

SysML v2 was created to overcome the limitations of SysML v1 and meet the needs of modern, complex, and digital systems engineering.

- **Clarify Semantics**: Reduce ambiguity and inconsistency in models.
- **Enable Automation**: Support machine-readable, executable models.
- **Improve Expressiveness**: Better model structure, behavior, constraints, and parametrics.
- **Enhance Usability**: More intuitive, flexible modeling structures for engineers.
- **Support Interoperability**: Easier exchange and integration across tools and domains.

2001	INCOSE and OMG initiate need for SysML modeling language
2003	SysML Partners formed; SysML v1 development bejns
2006	SysML v1.0 published by OMG
2012–2017	Experience with SysML v1 points out limitations
2017	SysML v2 Request for Proposal (RFP) issued
2021	SysML v2 public draft released for review
2023	SysML v2 Finalization Task Force (FTF) working toward OMG adoption
2024	SysML v2 formally released as OMGstandard

After learning what SysML v2 is, it's important to ask: **Why was SysML v2 needed? Why didn't we just keep using SysML v1?**

The answer is simple: **The engineering world changed.** And the tools needed to support it had to evolve as well.

First, SysML v1, while a great advancement for its time, had **ambiguities in its semantics**: Different modelers and different tools could interpret the same diagrams differently. This led to inconsistencies and limited confidence in automated analysis or model exchange. **SysML v2 clarifies the semantics** dramatically:

- Every element, relationship, and behavior is precisely defined.
- Models are not just visual artifacts — they become machine-readable, analyzable system representations.

Second, modern systems demand **automation**: Systems Engineers need to simulate, verify, trace, and validate systems continuously — long before physical hardware is built. SysML v2 supports **executable models** — models that can be reasoned about by tools for early simulation and checking.

Third, complex modern systems — Smart cities, autonomous drones, integrated healthcare platforms — Demand **greater modeling expressiveness**:

- More powerful ways to describe dynamic behavior.
- Stronger constraint handling for parametric analysis and trade studies.
- Richer modeling of interaction scenarios and interfaces.

Fourth, SysML v2 focuses on **usability improvements**: SysML v2 re-structures modeling concepts to be more intuitive. Engineers can focus more on the system they are building, and less on wrestling with language limitations.

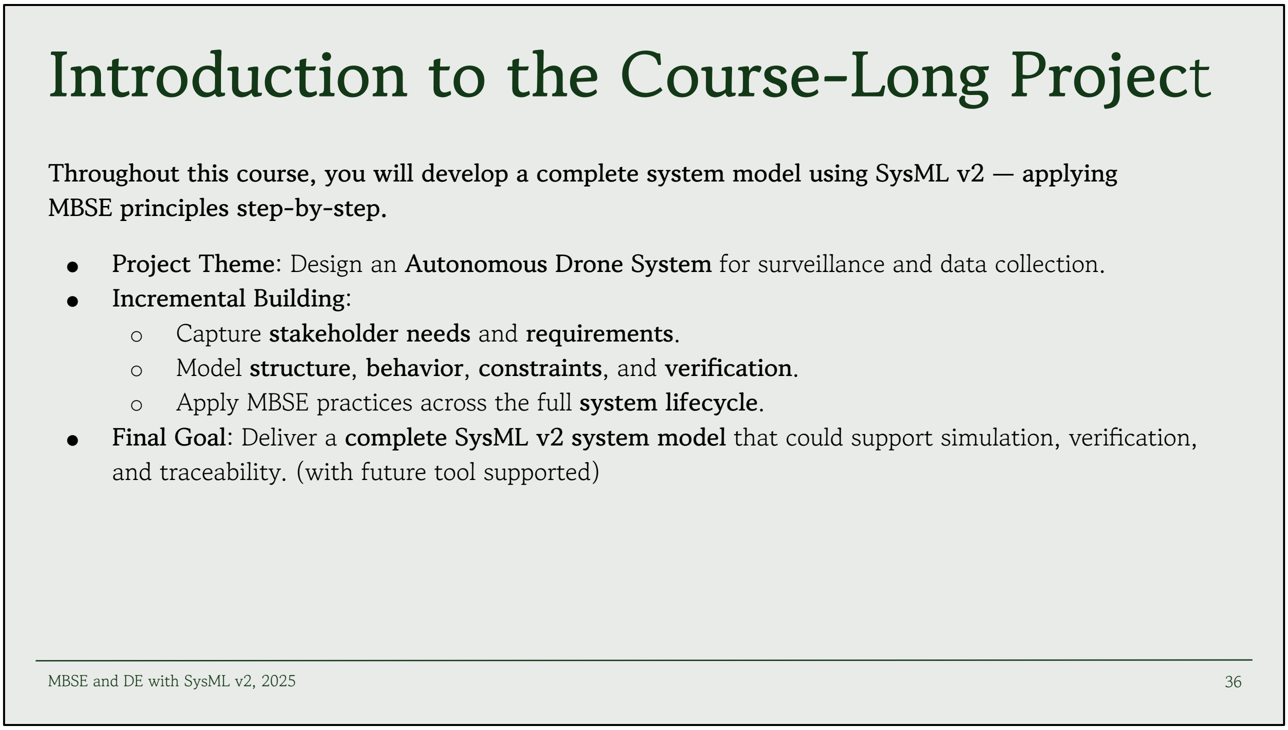
Finally, in a world where systems are developed collaboratively across companies, countries, and tools, **SysML v2 improves interoperability**:

- Standardized data exchange formats.
- Consistent modeling semantics.
- Better ability to integrate with domain-specific models (software, mechanical, electrical).

In short: **SysML v2 was not just an upgrade — it was a strategic redesign** to support the future of engineering:

- Larger systems.
- Faster development.
- Deeper simulation and verification.
- Greater collaboration.
- Realization of true Digital Engineering.

Without SysML v2, MBSE could not scale to meet the challenges of the 21st century.



Introduction to the Course-Long Project

Throughout this course, you will develop a complete system model using SysML v2 — applying MBSE principles step-by-step.

- **Project Theme**: Design an **Autonomous Drone System** for surveillance and data collection.
- **Incremental Building**:
  - Capture **stakeholder needs** and **requirements**.
  - Model **structure**, **behavior**, **constraints**, and **verification**.
  - Apply MBSE practices across the full **system lifecycle**.
- **Final Goal**: Deliver a **complete SysML v2 system model** that could support simulation, verification, and traceability. (with future tool supported)

To anchor everything we learn into a practical experience, this course includes a **course-long project** where you will step-by-step **build a full system model** using SysML v2.
The project theme will be: **Designing an Autonomous Drone System** focused on **surveillance and data collection** missions.
This project mirrors real-world complexity in a manageable way — allowing you to experience capturing needs, defining architecture, modeling behavior, managing requirements, and setting up verification strategies — just like Systems Engineers do on real aerospace or defense projects.
The project will be built **incrementally**:

- First, you'll **identify stakeholders** — military users, maintenance crews, command centers — and capture their **needs and concerns**.
- Next, you'll **refine needs into system requirements**, separating functional from non-functional requirements.
- Then, you'll build the **system structure** — defining major subsystems like sensors, navigation, communications, and power.You'll model **system behaviors** — such as mission profiles, emergency procedures, and autonomous decision-making.
- You'll define **constraints and parameters** — like endurance limits, communication ranges, and payload capacities.
- You'll establish **verification and validation models** — showing how you would test and confirm that the drone meets its requirements.
- Along the way, you'll learn to use **traceability** — connecting needs to requirements, structure to behavior, and requirements to verification.

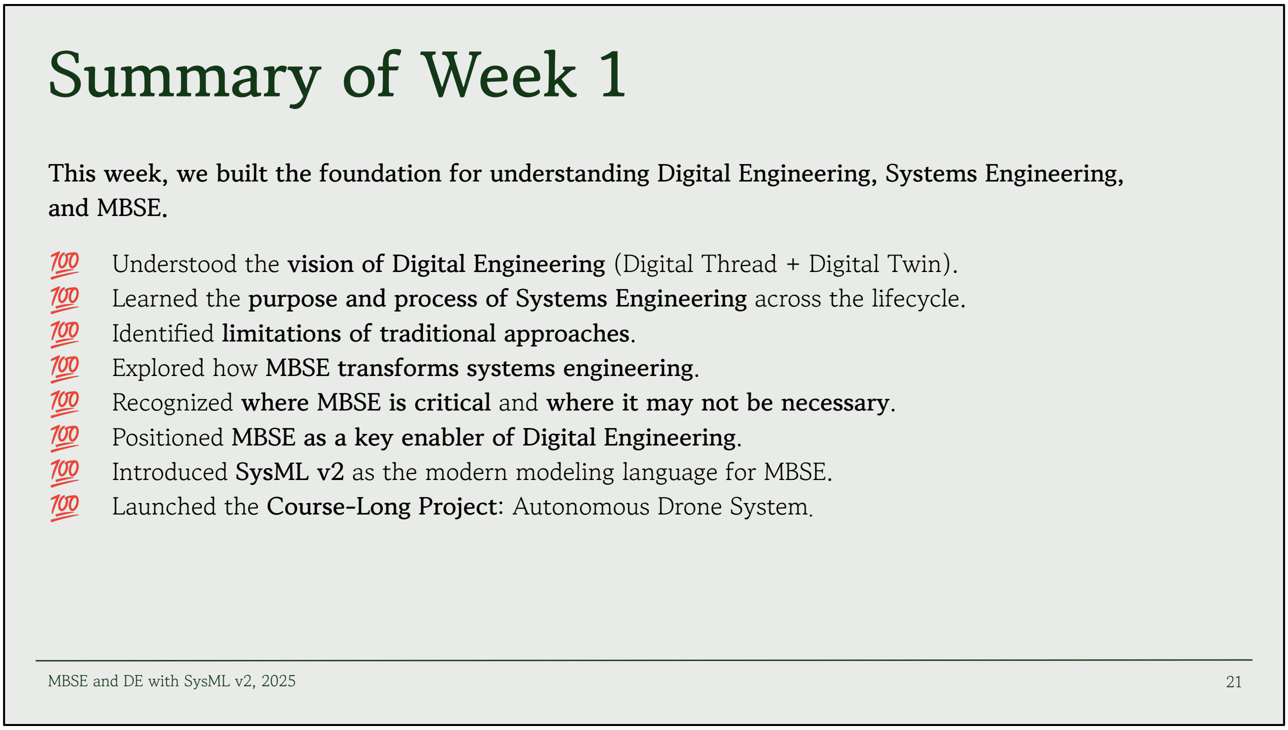
The final goal: At the end of the course, each student (or team) will deliver a **complete, coherent SysML v2 system model** — a model that could realistically support simulation, analysis, collaboration, and even early-stage Digital Twin creation.

This project will be **your showcase of MBSE skills**:

- Applying theory to practice.
- Thinking across the full lifecycle.
- Using SysML v2 properly and professionally.

Every major topic we learn each week will directly advance your system model. So learning and doing will always stay connected.

By the end of this course, you will not only understand MBSE and SysML v2 — You will have **applied them to a real system modeling challenge**, preparing you for professional systems engineering work.



Summary of Week 1

This week, we built the foundation for understanding Digital Engineering, Systems Engineering, and MBSE.

💯 Understood the **vision of Digital Engineering** (Digital Thread + Digital Twin).
💯 Learned the **purpose and process of Systems Engineering** across the lifecycle.
💯 Identified **limitations of traditional approaches**.
💯 Explored how **MBSE transforms systems engineering**.
💯 Recognized **where MBSE is critical** and **where it may not be necessary**.
💯 Positioned **MBSE as a key enabler of Digital Engineering**.
💯 Introduced **SysML v2** as the modern modeling language for MBSE.
💯 Launched the **Course-Long Project**: Autonomous Drone System.

This week, we have built a strong foundation for everything we will learn and practice in this course.

First, we explored the **vision of Digital Engineering** — how connected digital models transform systems development across the entire lifecycle, enabled by concepts like the **Digital Thread** and **Digital Twin**.

We learned **what Systems Engineering is** — a disciplined, lifecycle-focused approach to ensuring complex systems succeed — and understood the official **INCOSE Systems Engineering Lifecycle**.

We discussed the **limitations of traditional document-based Systems Engineering**, and saw **how MBSE addresses these challenges** by using integrated models.

We also recognized that MBSE is a **strategic tool**, best applied **where complexity, risk, or lifecycle needs demand it** — and not necessarily everywhere.

Importantly, we positioned MBSE properly inside **Digital Engineering**: MBSE anchors the Digital Thread and supports Digital Twin creation — making the digital transformation of systems possible.

We introduced **SysML v2**, the new, powerful modeling language we will use, and explained **why it was developed** to meet modern engineering challenges.

Finally, we launched your **Course-Long Project**:

- A practical, hands-on system modeling experience —
- Building an Autonomous Drone System model —
- Applying MBSE principles step-by-step every week.

By mastering these foundations, you are now ready to dive deeper into the real practice of MBSE — modeling the system lifecycle from concept to verification, using SysML v2.

QUESTION!