

Improving the Modeling Experience with Gen-AI

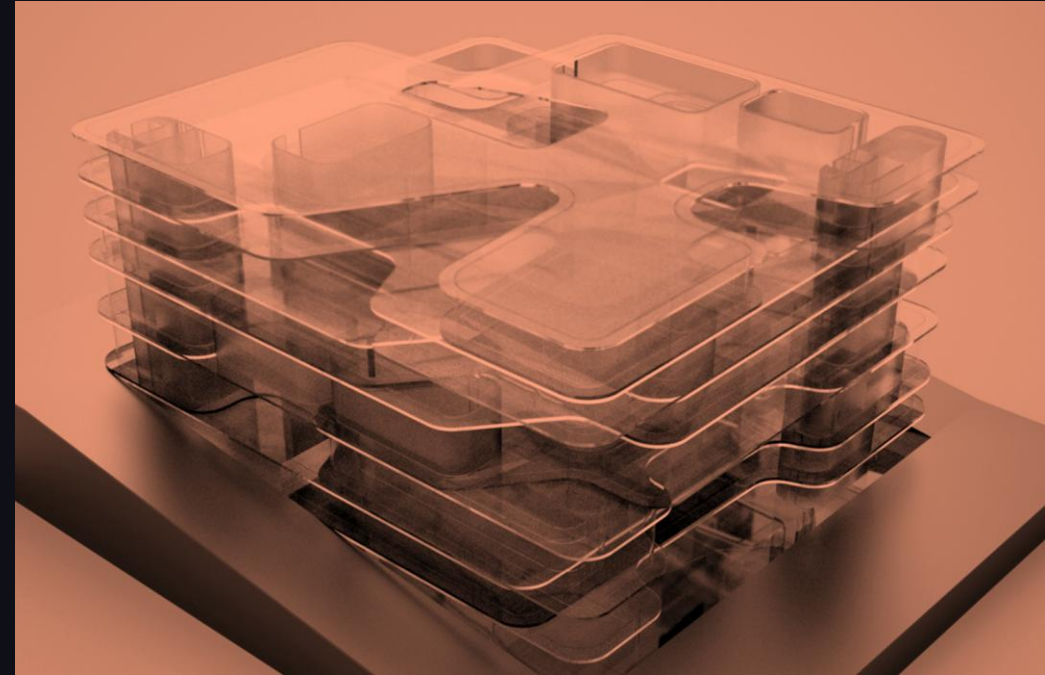
Pedro J. Molina, PhD. / Metadev

Pablo Rodríguez / Metadev

Improving the Modeling Experience with Gen-AI

Pedro J. Molina, PhD. / Metadev

Pablo Rodríguez / Metadev



Content

- The Abstraction Ladder
- Modeling tools & LLMs
- Opportunities for AI integration
- Places where not to apply AI
- MDE: The value was always in the model
- Machines does not need diagrams
- Humans in control, machines implementing the rest
- Exponential times



The Abstraction Ladder

*"The history of **Software Engineering** is the one of raising the **abstraction levels**."*

Grady Booch.



Layer 5. Natural Language

Layer 4. Formal Specifications

Layer 3. Programing Languages

Layer 2. Assembler

Layer 1. Machine Code

$$\begin{bmatrix} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 \\ a_7 & a_8 & a_9 \end{bmatrix} \begin{bmatrix} b_1 & b_2 & b_3 \\ b_4 & b_5 & b_6 \\ b_7 & b_8 & b_9 \end{bmatrix} = \begin{bmatrix} c_1 & c_2 & c_3 \\ c_4 & c_5 & c_6 \\ c_7 & c_8 & c_9 \end{bmatrix}$$

LLMs aka *the Stochastic Parrots*

Pros:

- Compile great amount of knowledge
- Good at predicting... the next token
- Great at summarizing information
- Brute force approaches humans cannot do
- Cheap hard workers

Cons:

- Probabilistic: Non determinist → Non Repeatable
- Limited innovation (no think out of the box) (only remix and combine)
- Cannot take accountability, liability, responsibility...

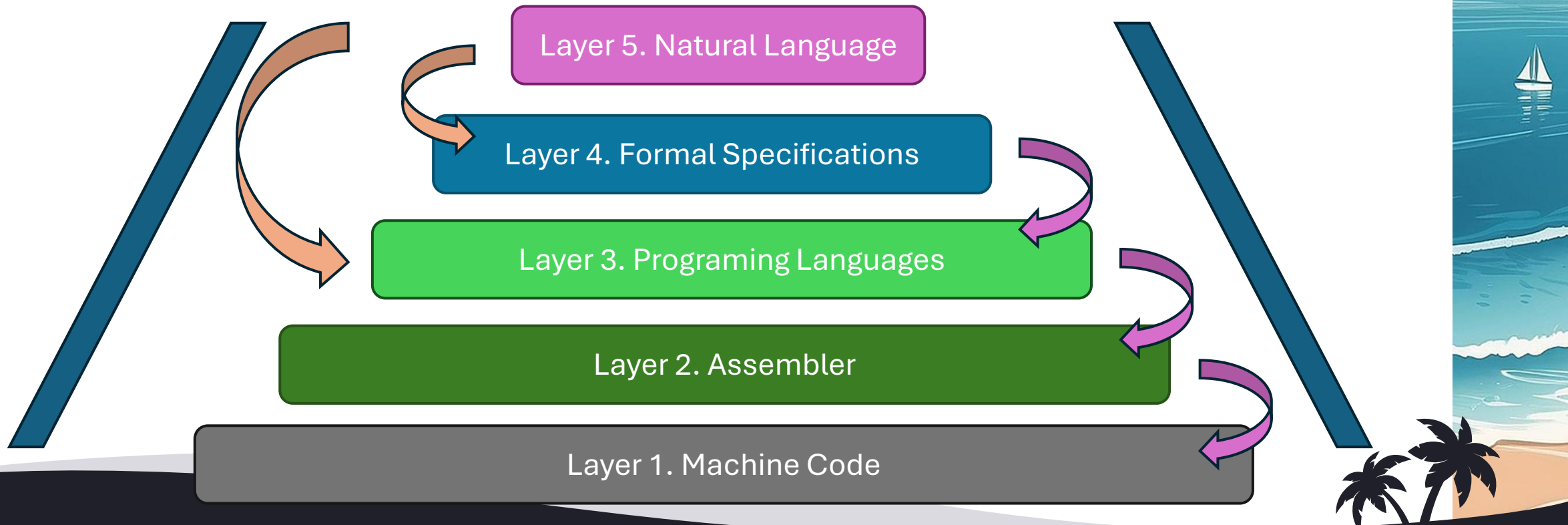


The Abstraction Ladder Revisited

Two paths to move throw the ladder:

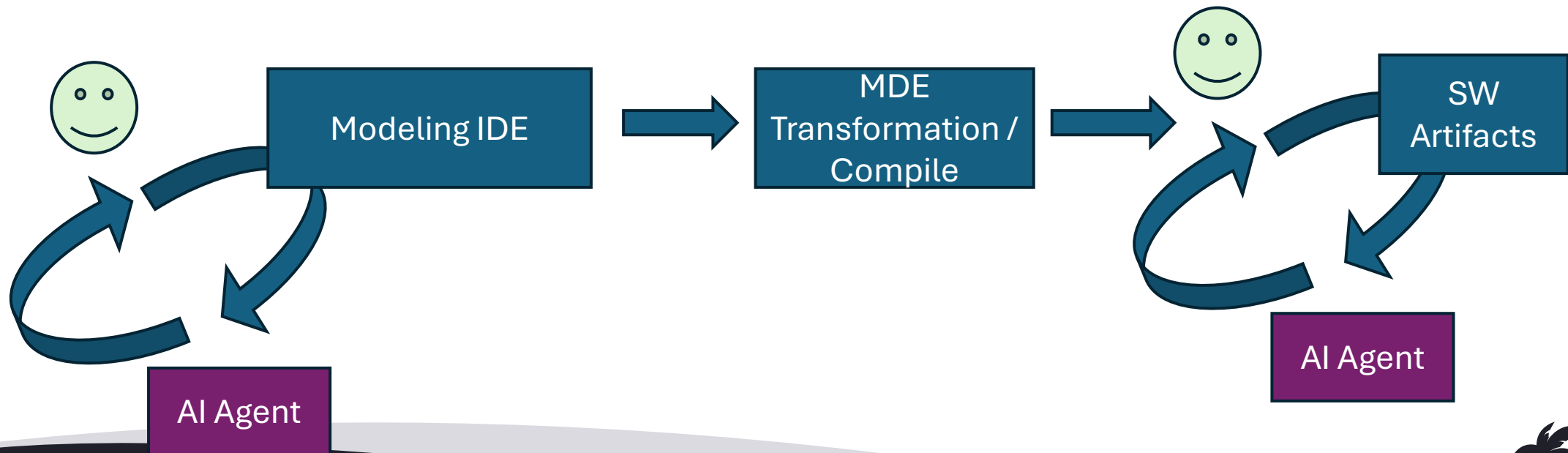
Probabilistic Slope

Deterministic Slope

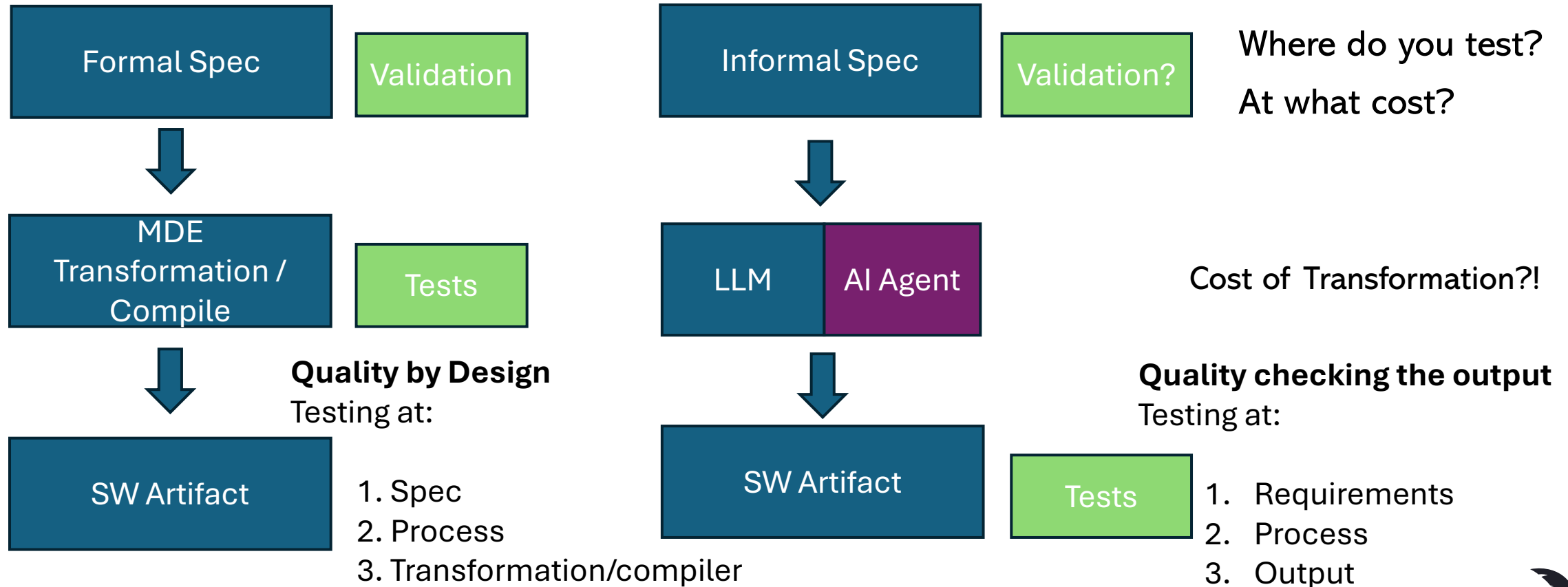


Enhanced MDE Design Flow

- AI as an **exoskeleton** to enhance the Engineer.
- Determinist preserve properties, semantics and qualities.
- Probabilistic outcomes are not necessarily bad, but requires more testing, validation and quality assurance.



Deterministic vs Probabilistic Flows



The Pareto Rule (once again)

- What if **80%** of your system could be delivered in a **deterministic way**?
- What if you complete the rest with **probabilistic methods** plus convergence criteria?
- Are we improving the time-to-market, quality and cost of the full system?



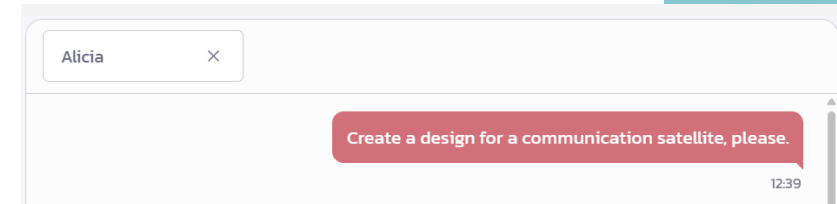
Demos

- Structura <https://structura.tools>
 - Voice supported assistant
 - Model assistant
 - MCP
- Apricot <https://apricot.tools>
 - SysML2 specialized AI
 - MCP



Opportunities for LLMs integration in modeling environments

- Assistants
- Voice input
- Modeling agents
- Model validation
- MCPs: modeling tooling as tools for agents
- RAG with curated models and domain specific information
- AI experts in niche domain modeling: SW architect, testing, security
- Template extraction/generalization for code generation
- Agentic UI



Opportunities for LLMs integration in modeling environments

The screenshot displays a SysML modeling environment with three main panels:

- Explorer:** A tree view on the left showing the project structure. It includes a package `CommSatellite` with several elements:
 - `ScalarValues` (import)
 - `ISQ` (import)
 - `OrbitKind` (enum) with items: `LEO`, `MEO`, `GEO`, `HEO`.
 - `StatusKind` (enum) with items: `nominal`, `degraded`, `safe_mode`, `off`.
 - `PolarizationKind` (enum) with items: `linear`, `circular`, `dual`.
 - Attributes: `Frequency`, `PowerWatts`, `TempKelvin`, `DataRateBps`, `MassKg`, `AngleDeg`.
 - Ports: `RFPort` (with attributes `centerFreq`, `bandwidth`, `polarization` and items `rxSignal`, `txSignal`), `PowerPort` (with items `powerIn`, `powerOut`), and `DataPort`.
- Code:** A central panel showing the SysML model's internal structure in a code-like format. It defines the `CommSatellite` package, imports, enumerations, value types, and port definitions.
- Diagram:** A panel on the right showing a graphical representation of the model, though it is mostly obscured by the code view.

```
1 package CommSatellite {
2   private import ScalarValues::*;
3   private import ISQ::*;
4
5   // =====
6   // Enumerations
7   // =====
8
9   enum def OrbitKind { LEO; MEO; GEO; HEO; }
10  enum def StatusKind { nominal; degraded; safe_mode; off; }
11  enum def PolarizationKind { linear; circular; dual; }
12
13  // =====
14  // Value types
15  // =====
16
17  attribute def Frequency    >> Real; // Hz
18  attribute def PowerWatts   >> Real; // W
19  attribute def TempKelvin   >> Real; // K
20  attribute def DataRateBps  >> Real; // bit/s
21  attribute def MassKg       >> Real; // kg
22  attribute def AngleDeg     >> Real; // degrees
23
24  // =====
25  // Port definitions
26  // =====
27
28  port def RFPort {
29    attribute centerFreq : Frequency;
30    attribute bandwidth  : Frequency;
31    attribute polarization : PolarizationKind;
32    in item rxSignal : Real;
33    out item txSignal : Real;
34  }
35
36  port def PowerPort {
37    in item powerIn : Real;
38    out item powerOut : Real;
39  }
40
41  port def DataPort {
42    in item dataIn : Real;
```

The screenshot shows an AI assistant interface named "Alicia". At the top, a red button says "Create a design for a communication satellite, please." The time is 12:39. Below the button, the assistant provides a summary of a SysML model.

ALICIA

The model is valid and fully loaded. Here's a complete summary of what was designed:

Communication Satellite — SysML2 Model

Architecture Overview

The model is structured inside the `CommSat` package with the following layers:

Value Types & Vocabulary

Custom attribute definitions typed on `Real` cover all engineering quantities: `Mass_kg`, `Power_W`, `Temp_K`, `Freq_GHz`, `DeltaV_ms`, `EIRP_dBW`, etc. — providing semantic clarity and unit traceability.

Port Definitions

Five reusable port types establish the inter-subsystem interfaces:

Port	Signals
PowerPort	voltage, current
ThermalPort	heat load, temp sink
RFPort	RF power, frequency
DataPort	data rate
CommandPort	commands, telemetry

Sub-systems (8 total)

Subsystem	Key Parts	Notable Values
Structure	Primary/secondary structure	180 kg, Al honeycomb

Ask anything about your model...

English (US) · Balanced

Where **not** to apply AI

We know problems where a **deterministic** solution is already available

- Raw code generation → if probabilistic, not repeatable
- Ensuring **Compliance**
- Ensuring **Quality by Design**
- When output implies **Liability**



The value is in the model



The HOW:

- Technologies, stacks, languages and development frameworks evolve fast, get on trendy and some disappear with the same speed.

The WHAT:

- The model remains. The change frequency of the business rules are more stable than tech.

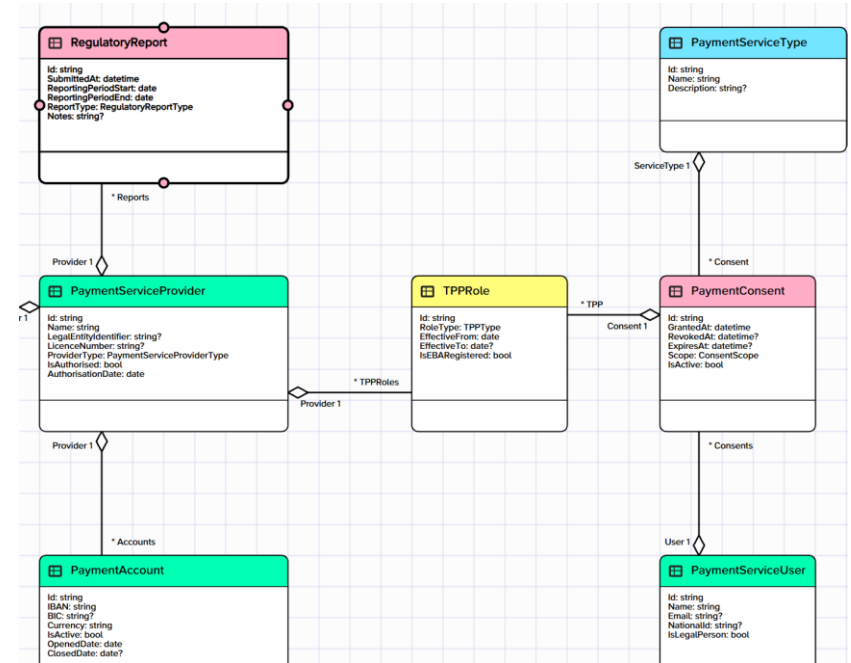


Machines do not need diagrams

Machines can use gigantic files with row JSON or YAML.

Humans do need models and diagrams

- to work at the relevant abstraction level
- to understand
- to review
- to take ownership and responsibility

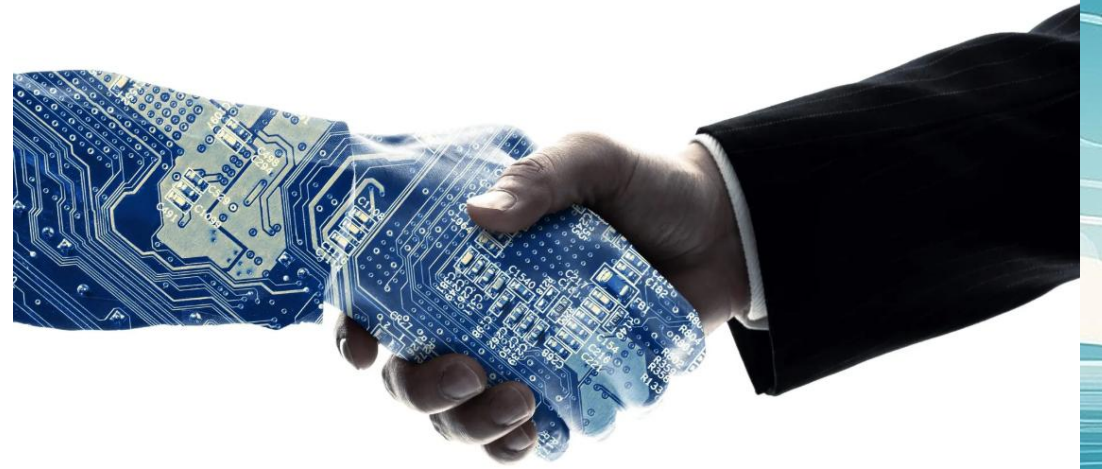


Humans in control

Curated models

- Understandable
- Maintainable
- Executable

Taking ownership, accountability and responsibility of them



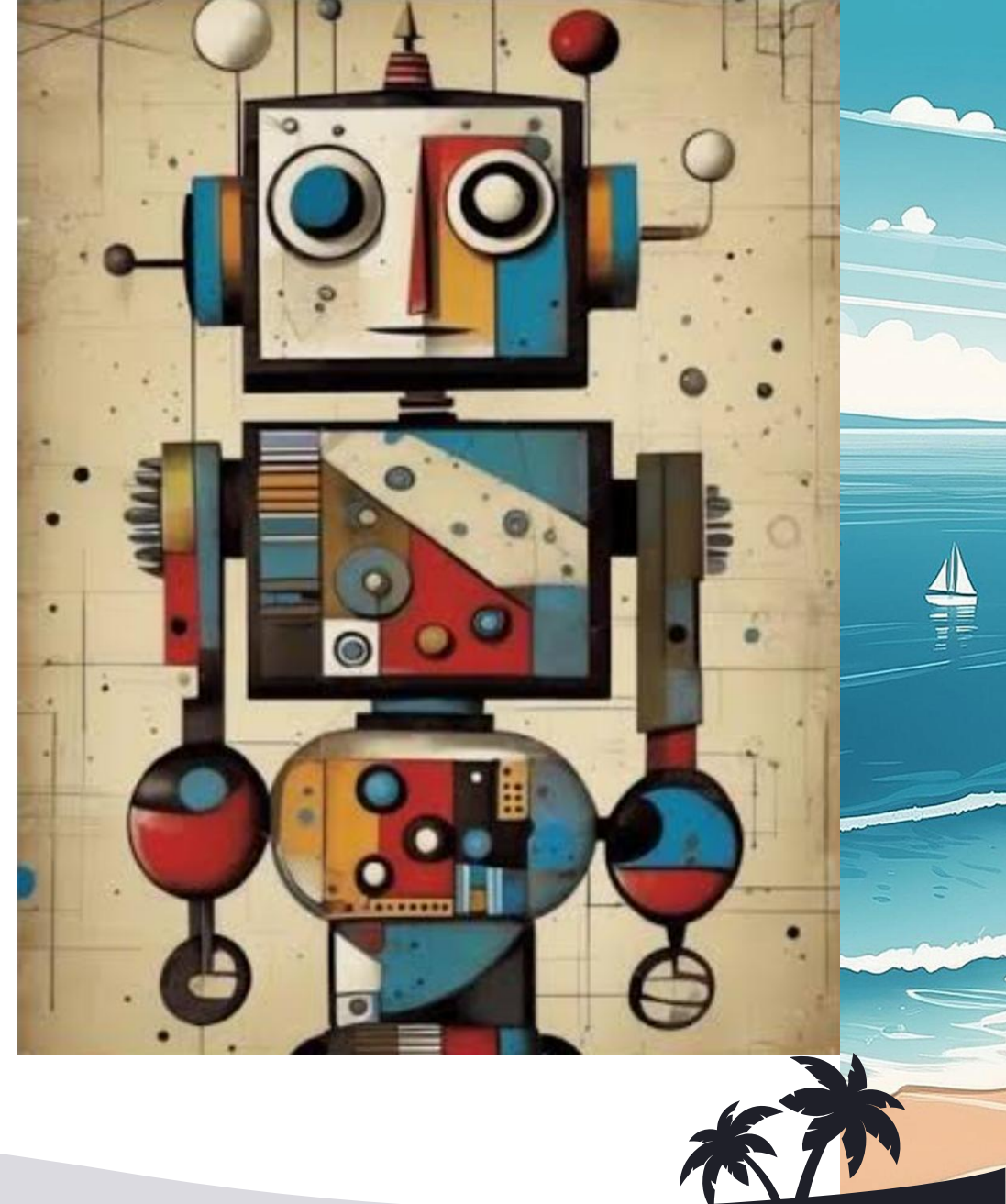
Machines (AI)

Executing the plans in the models

Machines (AI) needs feedback, criteria for evaluating its execution.

- Convergence criteria
- Stop criteria
- Kill criteria
- Quality Gates

Are we doing well?



Exponential times

AI improving impressively fast.

Engineers capable of **deliver much more** with better tools.

But they stay **probabilistic engines**, powerful stochastic parrots.

- Use them in scenarios where they excel
- Avoid using them for task we already know how to solve better or requires:
 - Accountability
 - Responsibility
 - Warranty
 - Liability
- When used, **insurance** the risk accordingly.





Questions?