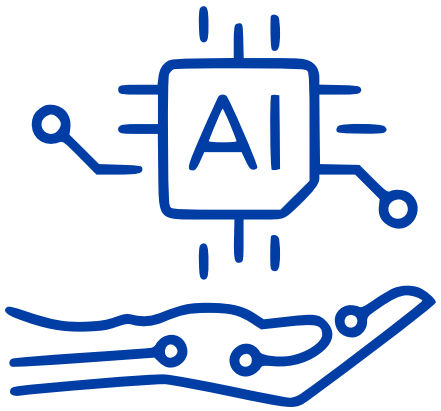
Details driving Innovation

AI-based Embedded Code Synthesis

AI Integration in Automotive Embedded Software Development

www.bitron.com

Agenda



- Project Overview
- AI System Architecture
- Evaluation Framework
- IT Infrastructure

www.bitron.com

Project Overview

Details driving Innovation

Introduction

Generative AI Integration in Automotive Embedded Development

PAIN POINTS

- Increasing complexity of automotive embedded software and growing number of requirements to be implemented
- Time-to-market pressure combined with rigorous A-SPICE processes and documentation compliance requirements
- General-purpose generative models produce demonstrative code that is not directly usable in industrial environments

SOLUTION

- Integration of generative AI into the existing development workflow without altering established processes
- Automatic generation of embedded software from structured requirements with human-in-the-loop validation
- Custom AI solution designed for real industrial automotive environments

Industry - Academia Partnership

Bitron | Politecnico di Milano

Collaboration between multidisciplinary academic research and industrial automotive expertise to develop concrete and industry-ready AI solutions

Project Scope

Where we integrate the AI Assistant into the Automotive V-Cycle

INTEGRATION STRATEGY

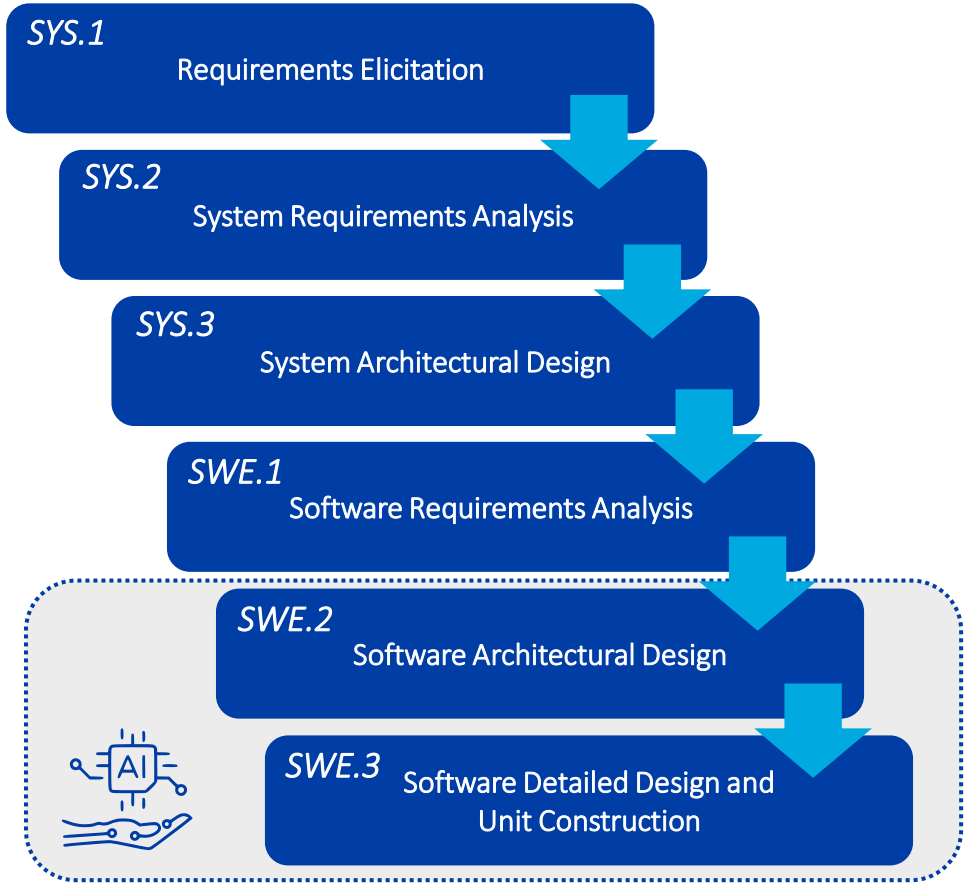
Not to replace the entire V-cycle, but to integrate an AI assistant into a specific development phase
Structured Requirements → Verified Embedded Code

DESIGN TRADE-OFF

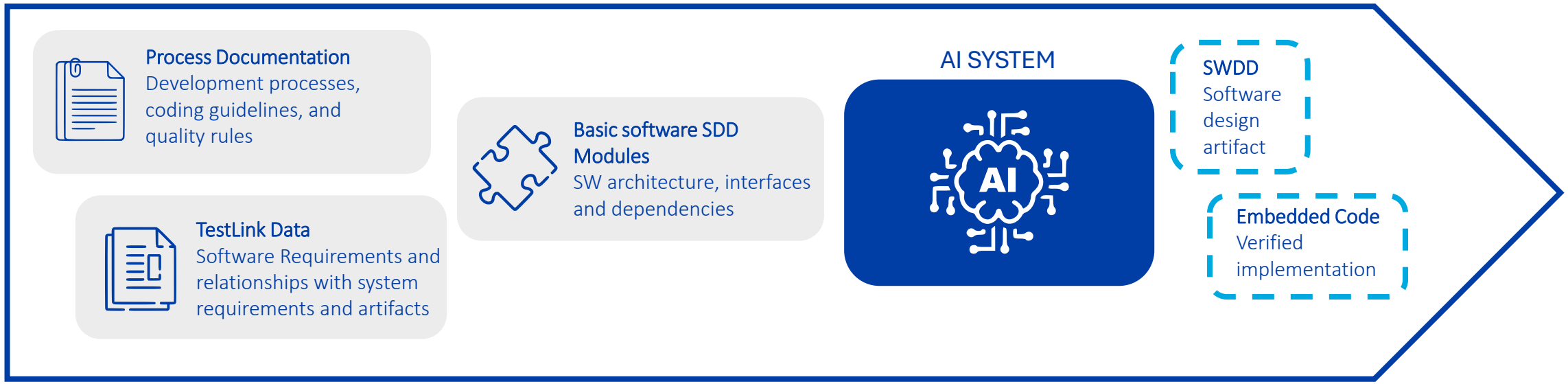
- **Human Control:** Fundamental requirement in automotive software development
- **Generative Capabilities:** acceleration of code synthesis from requirements

GUIDING PRINCIPLES

- **Human-in-the-loop:** AI as a support tool, with final human validation
- **Structured Input:** TestLink → refinement → LTL
- **Industrial RAG:** Legacy modules, A-SPICE, coding rules
- **Iterative Generation:** Pipeline multi-stage with feedback loop
- **Multi-level Validation:** Rule-based KPI, LLM as a Judge, Comparison against ground truth



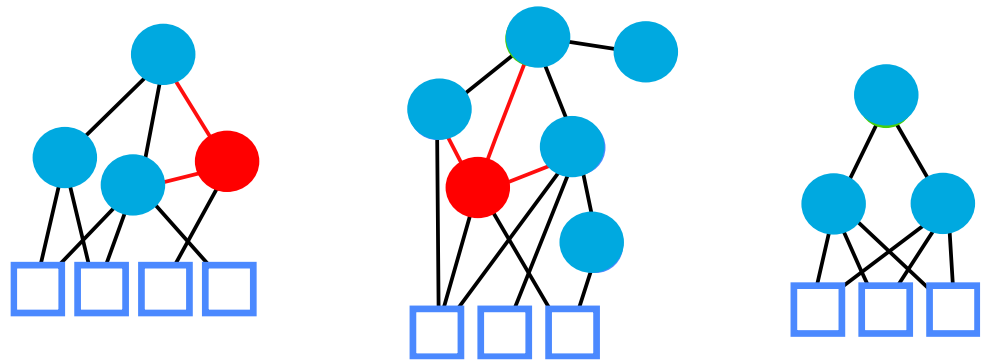
Data Sources and Requirements Clustering



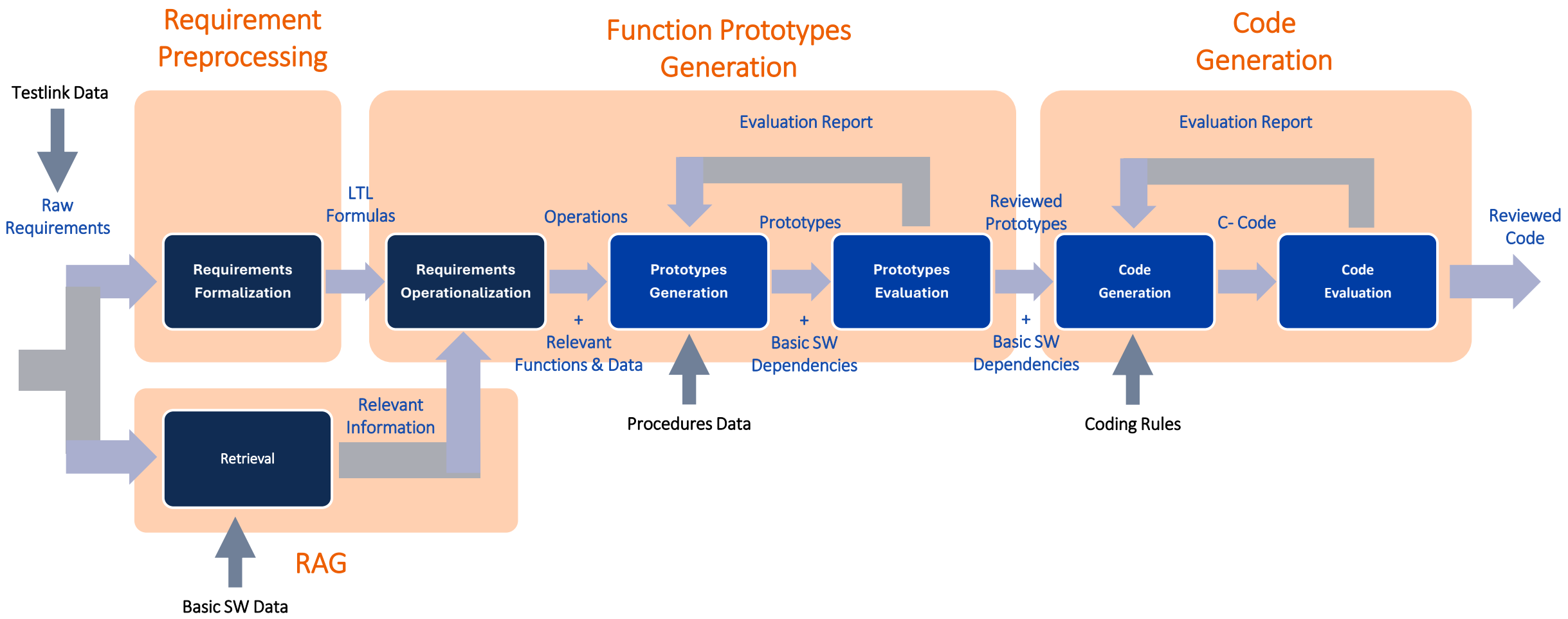
REQUIREMENTS CLUSTERING STRATEGY

Each cluster represents a coherent and relatively independent set of requirements to be implemented jointly.
Benefits:

- **Preserves the context:** Maintains dependencies and relationships between requirements
- **Reduces complexity:** Breaks down the problem into simpler subproblems
- **Improves scalability:** Makes generation more reliable and modular



Architecture Overview



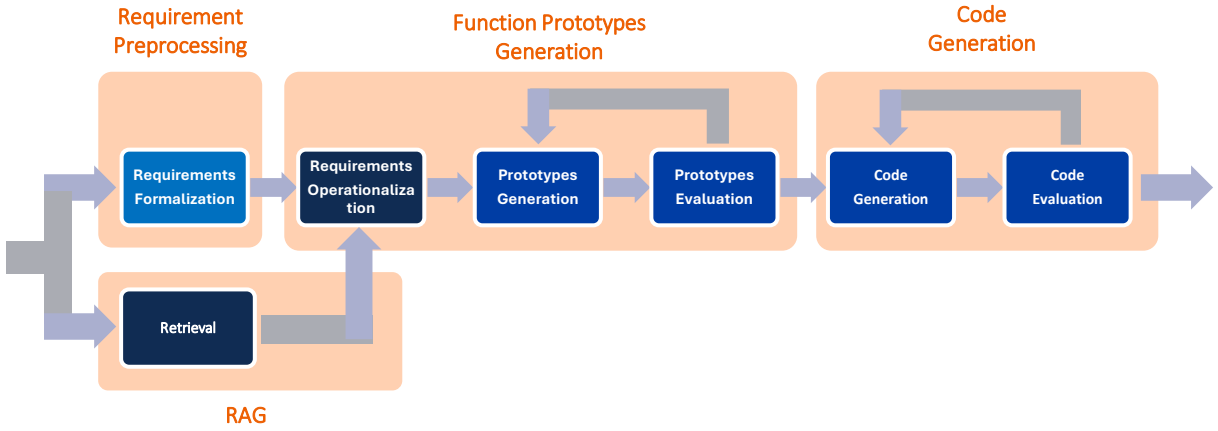
www.bitron.com

AI System Architecture

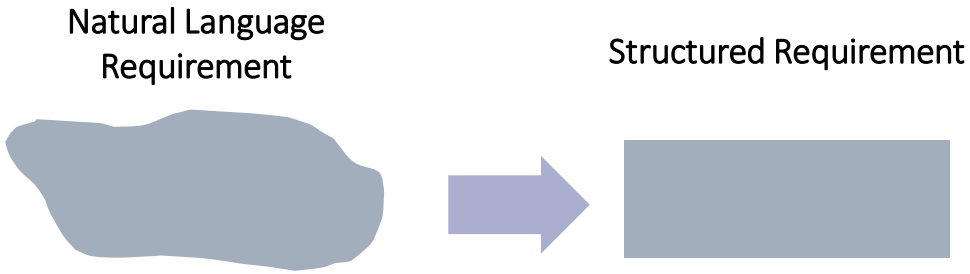
Details driving Innovation

Requirements Preprocessing

Why?



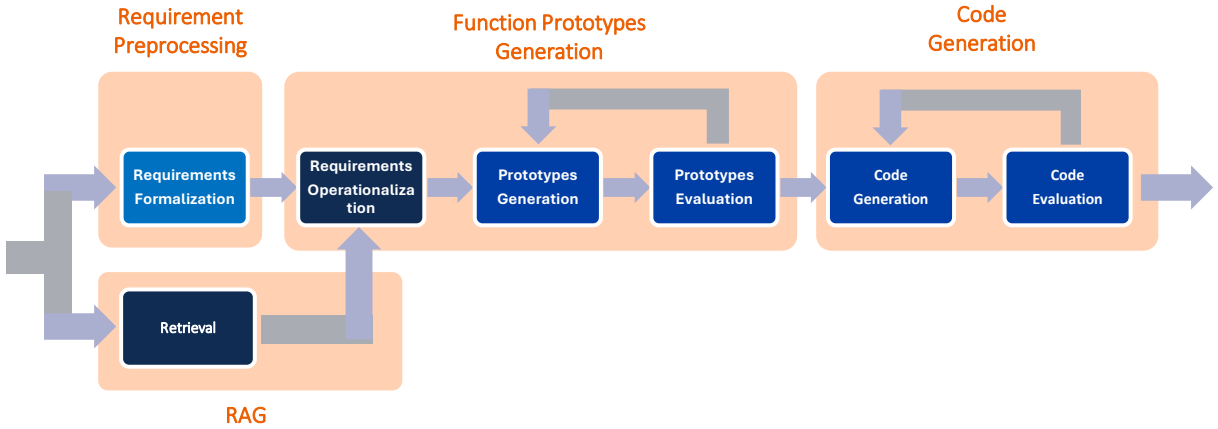
- Transform heterogeneous inputs into a **standard, structured and consistent format**.



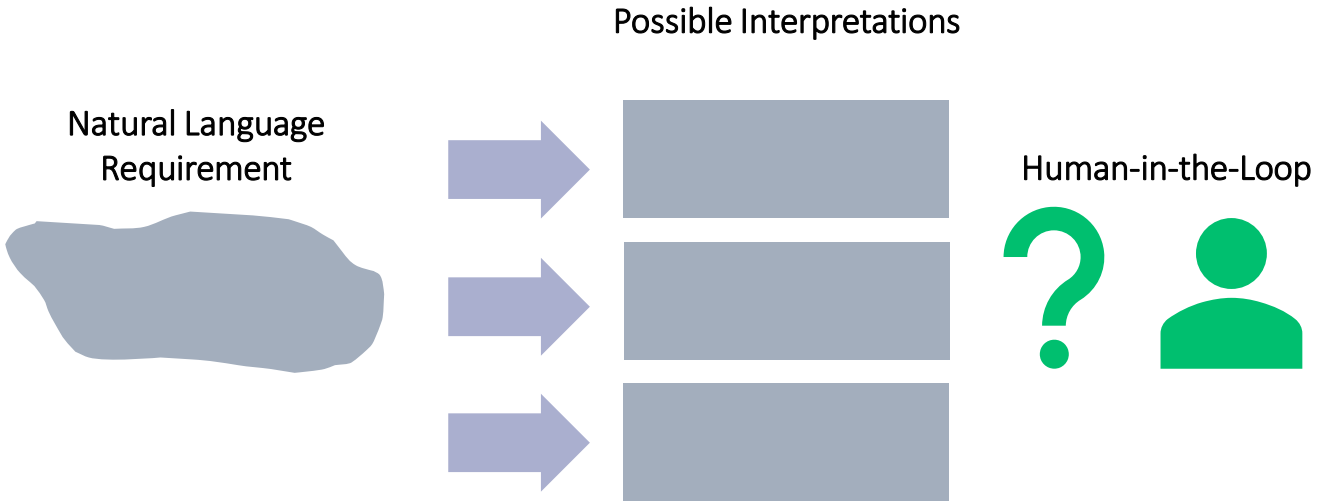
Natural Language Requirement:
If the button is kept pressed for the entire debounce time duration (T_d), then the system sends the 'Pressed' signal on the LIN.

Requirements Preprocessing

Why?



- Transform heterogeneous inputs into a **standard, structured and consistent format**.
- **Remove the intrinsic ambiguities** of natural language (through Human-in-the-Loop).

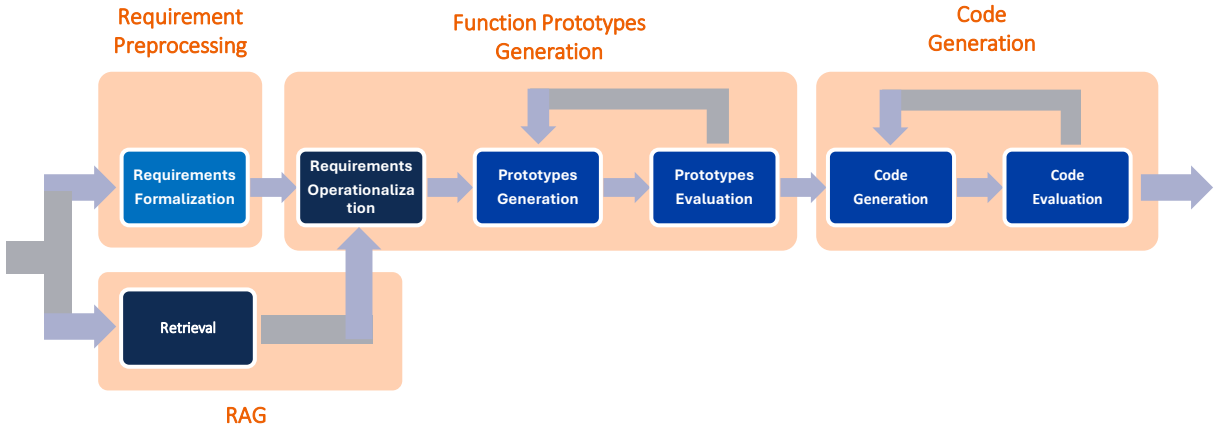


Natural Language Requirement:
If the button is kept pressed for the entire debounce time duration (T_d), then the system sends the 'Pressed' signal on the LIN.

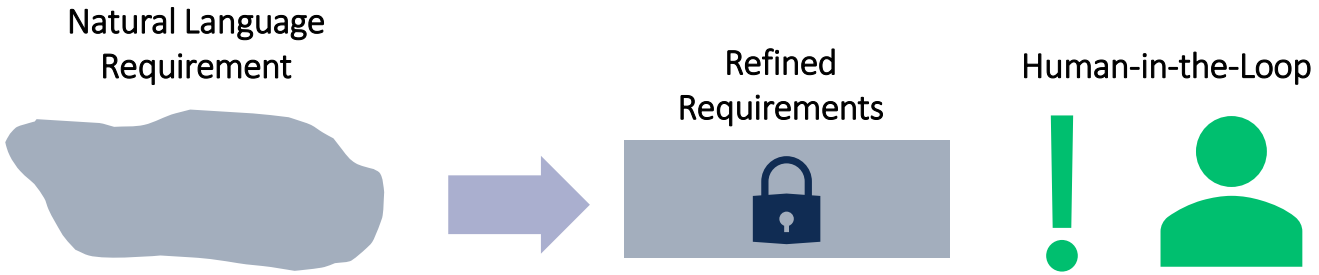
Interpretation 1: The system sends a **single pulse** at the exact moment when the press duration reaches T_d.
Interpretation 2: The system starts sending the signal and **keeps it active** until the button is released.

Requirements Preprocessing

Why?



- Transform heterogeneous inputs into a **standard, structured and consistent format**.
- **Remove the intrinsic ambiguities** of natural language (through Human-in-the-Loop).

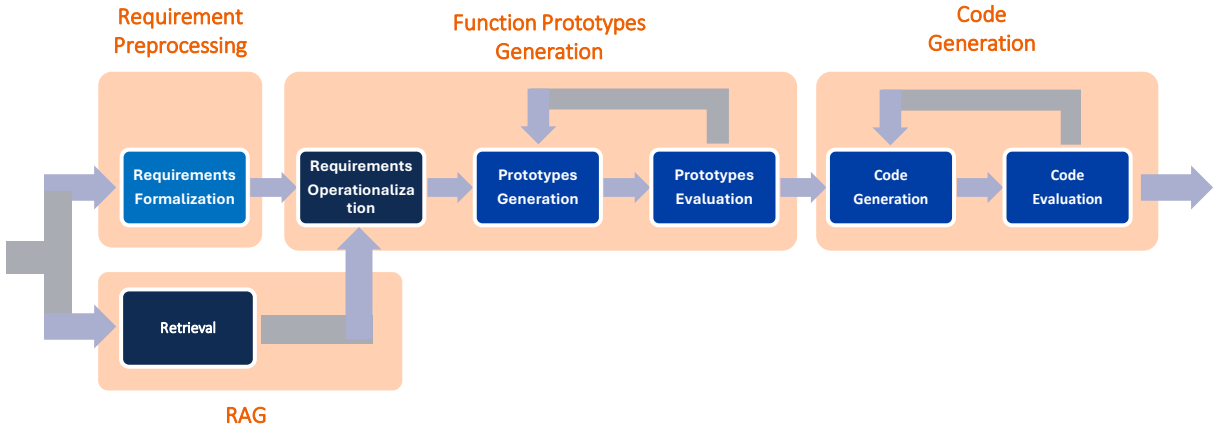


Natural Language Requirement:
If the button is kept pressed for the entire debounce time duration (T_d), then the system sends the 'Pressed' signal on the LIN.

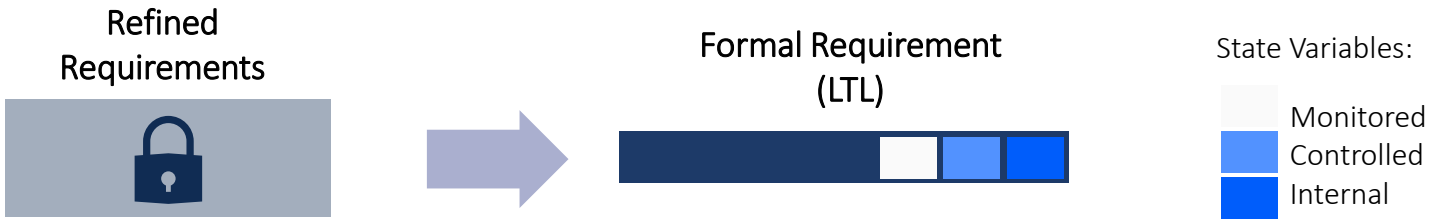
Interpretation 1: The system sends a **single pulse** at the exact moment when the press duration reaches T_d.
Interpretation 2: The system starts sending the signal and **keeps it active** until the button is released.

Requirements Preprocessing

Why?



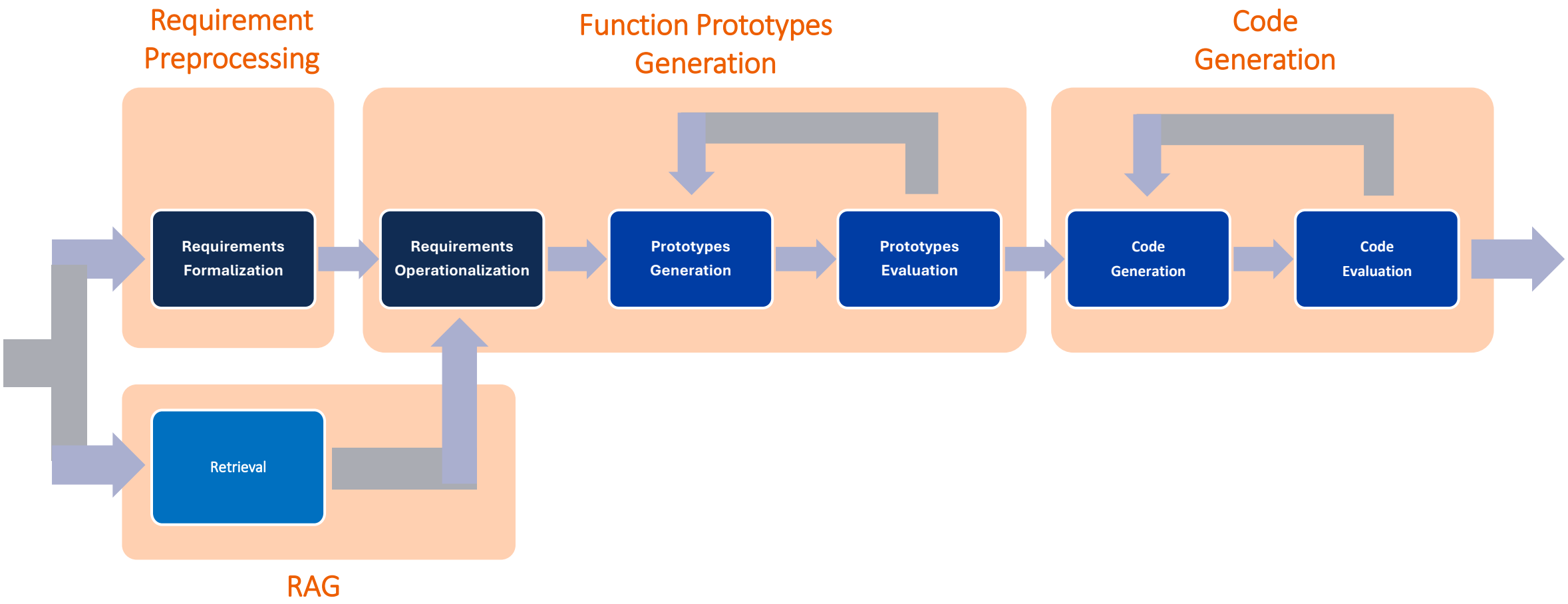
- Transform heterogeneous inputs into a **standard, structured, and consistent format**.
- **Remove the intrinsic ambiguities** of natural language (through Human-in-the-Loop).
- Translate the requirement into a **formal**, high-information-density formulation, useful for the rest of the pipeline.



Refined Requirement:
When the debounce timer expires, and the button has been LOW for the entire duration, send the (impulse) 'Pressed' signal to the LIN.

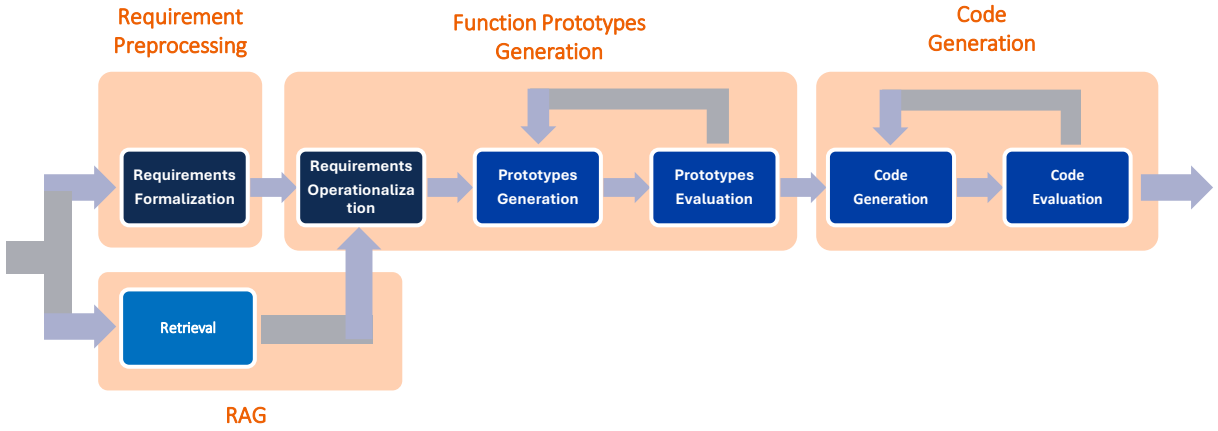
Linear Temporal Logic (LTL) Formula:
(DebounceTimer_Expired and ButtonLowThroughoutDebounce) → X(LIN_PutMessage_Pressed)
Monitored_Variable Monitored_Variable Controlled_Variable

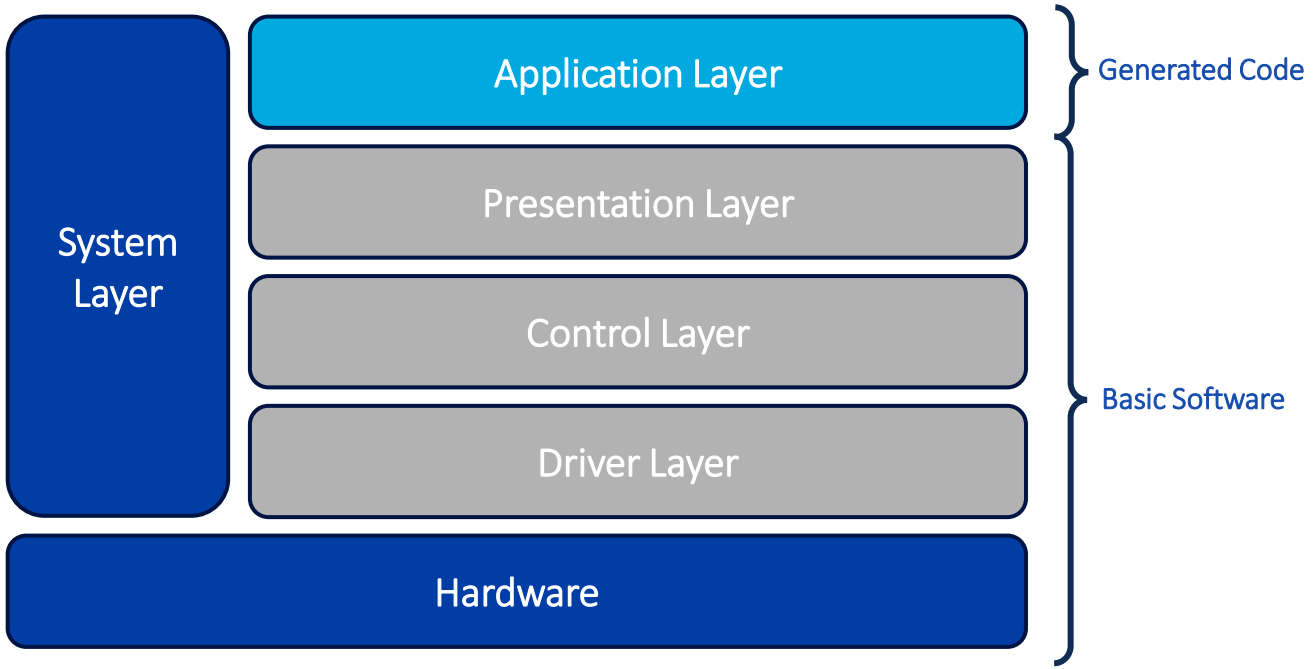
Retrieval Augumented Generation

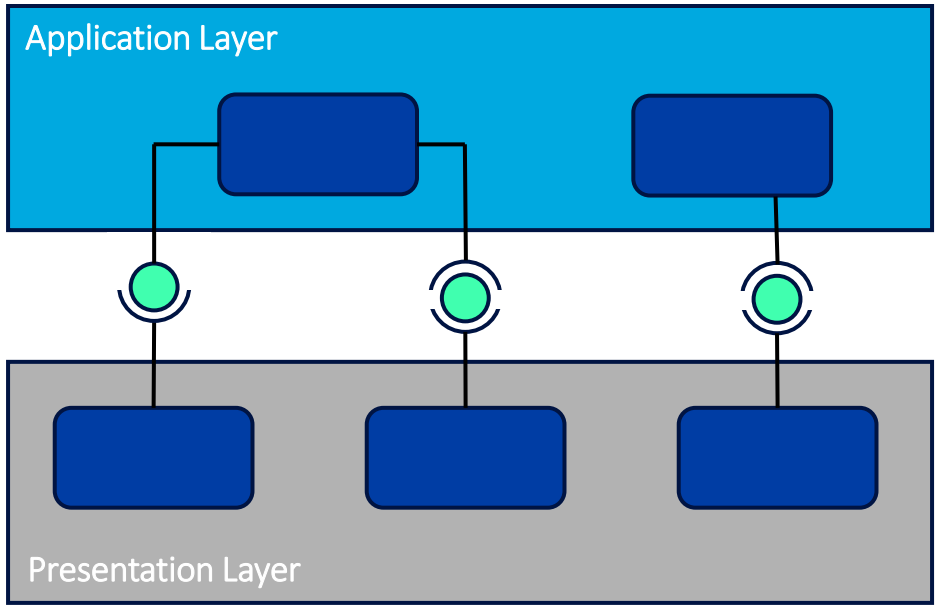


Retrieval Augmented Generation

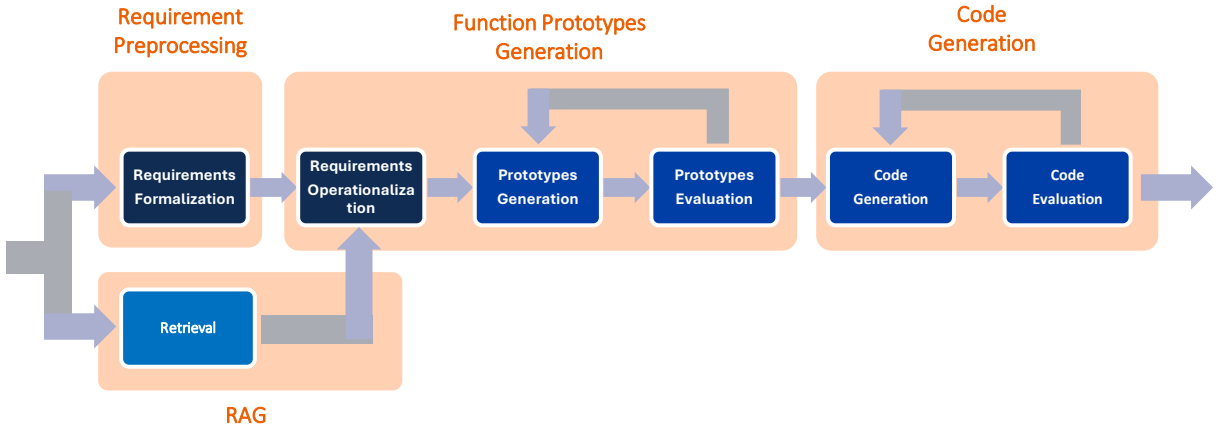
Why?

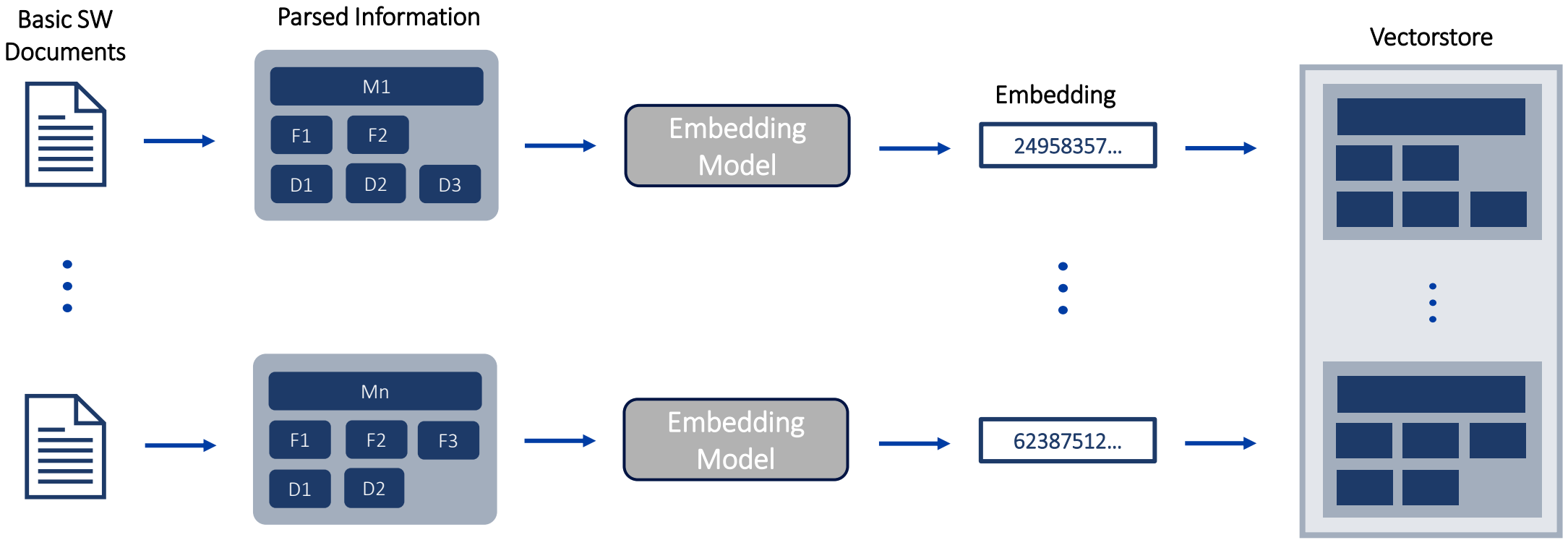




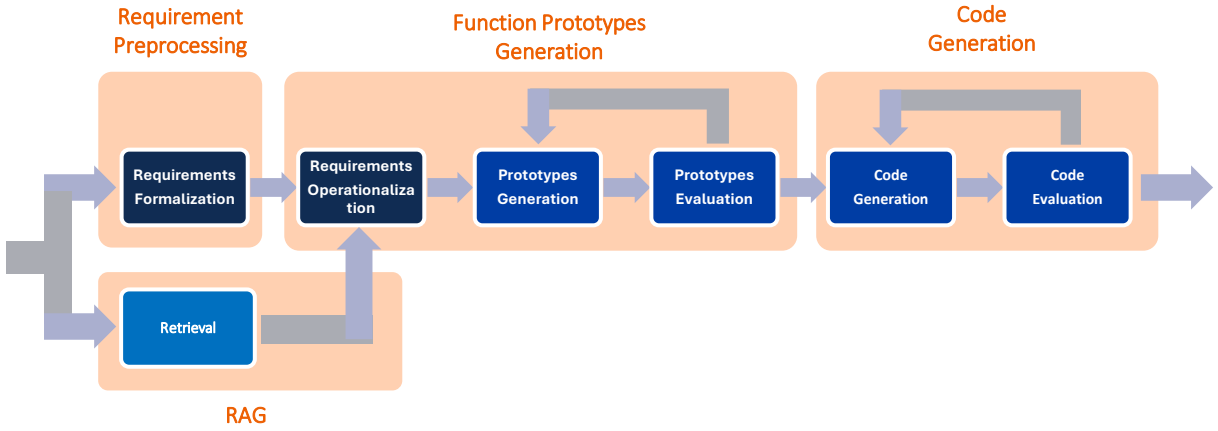


Vectorstore Creation



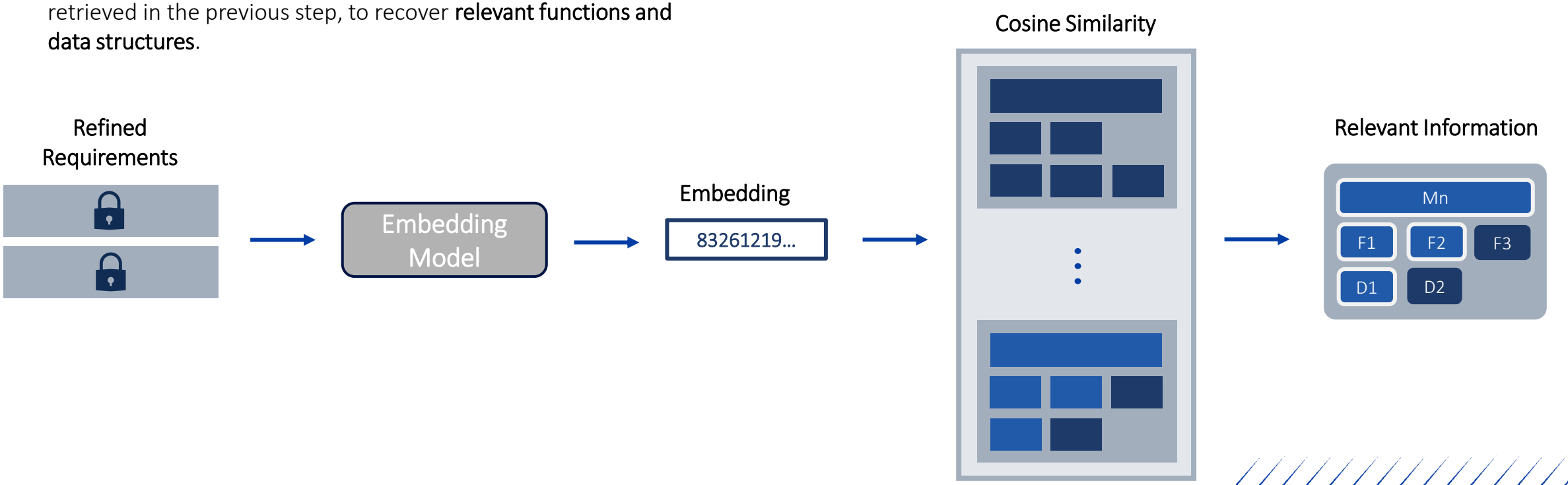


Retrieval Augmented Generation

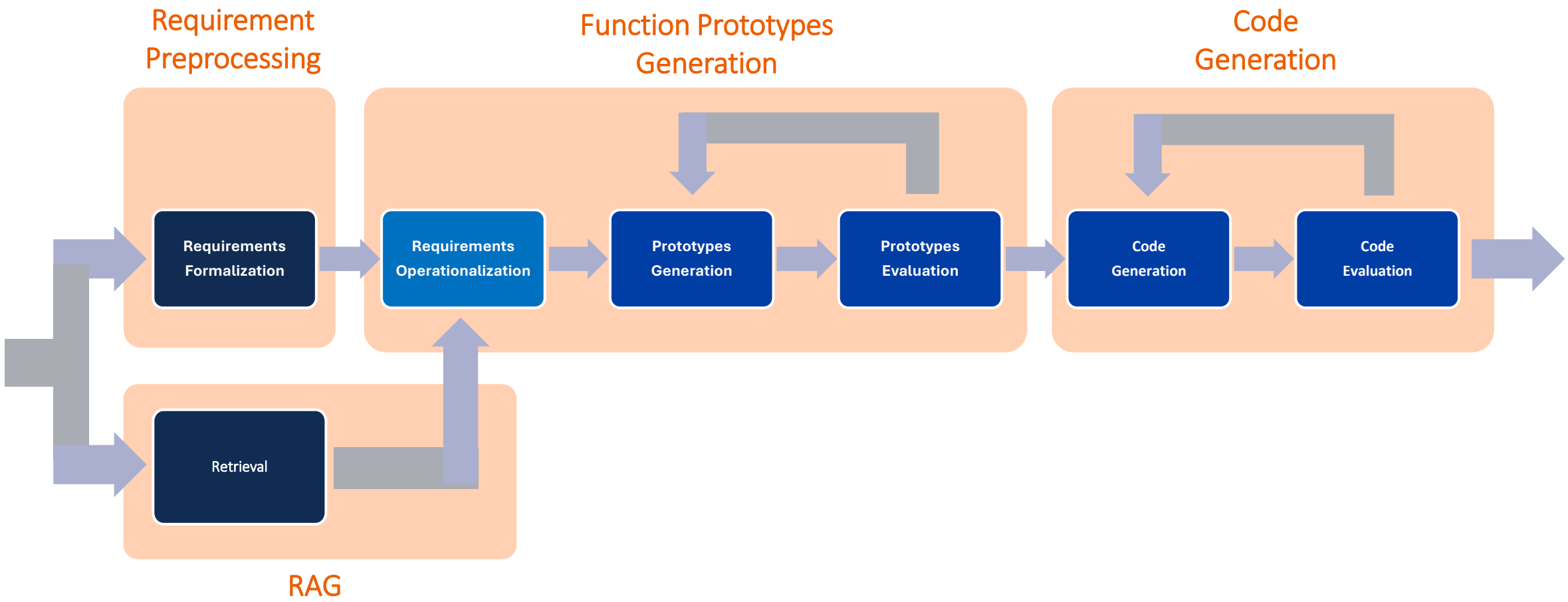


Adopted technique: **Hierarchical Retrieval**

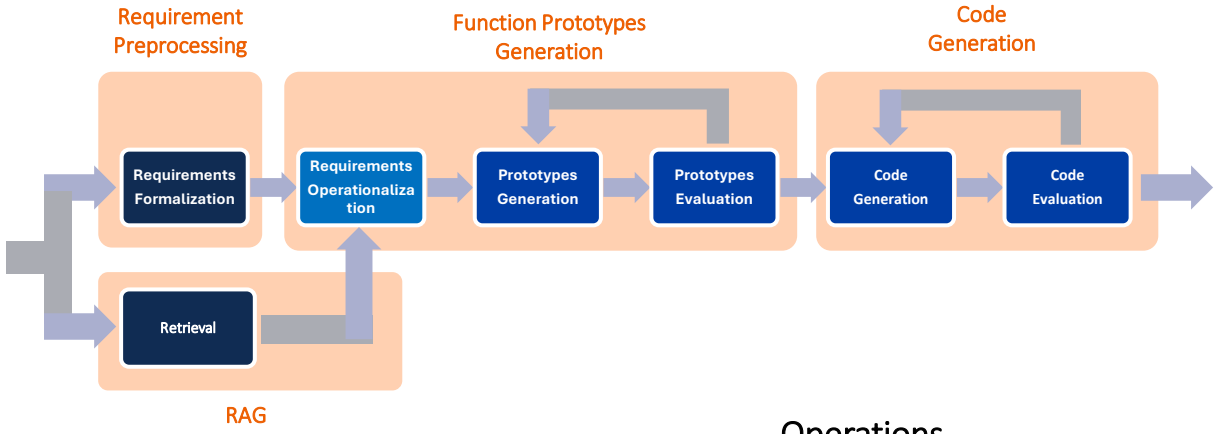
- First, we perform a similarity search on the filtered vector store (looking only at the **module descriptions**).
- Then, we perform a second **targeted retrieval** within the modules retrieved in the previous step, to recover **relevant functions and data structures**.



Requirement Operationalization



Requirement Operationalization



Formal Requirement (LTL)

Relevant Information

- Identify **state changes**
- Derive **domain preconditions / postconditions**
- Derive required **triggers**
- Derive **required preconditions / postconditions**
- Align operations with **BSW interfaces**

Operations

Relevant BSW Functions & Data

F2 D1

LTL Formula:(DebounceTimer_Expired and ButtonLowThroughoutDebounce) → X(LIN_PutMessage_Pressed)

Operation 1: Handle_DebounceTimer_Expiry

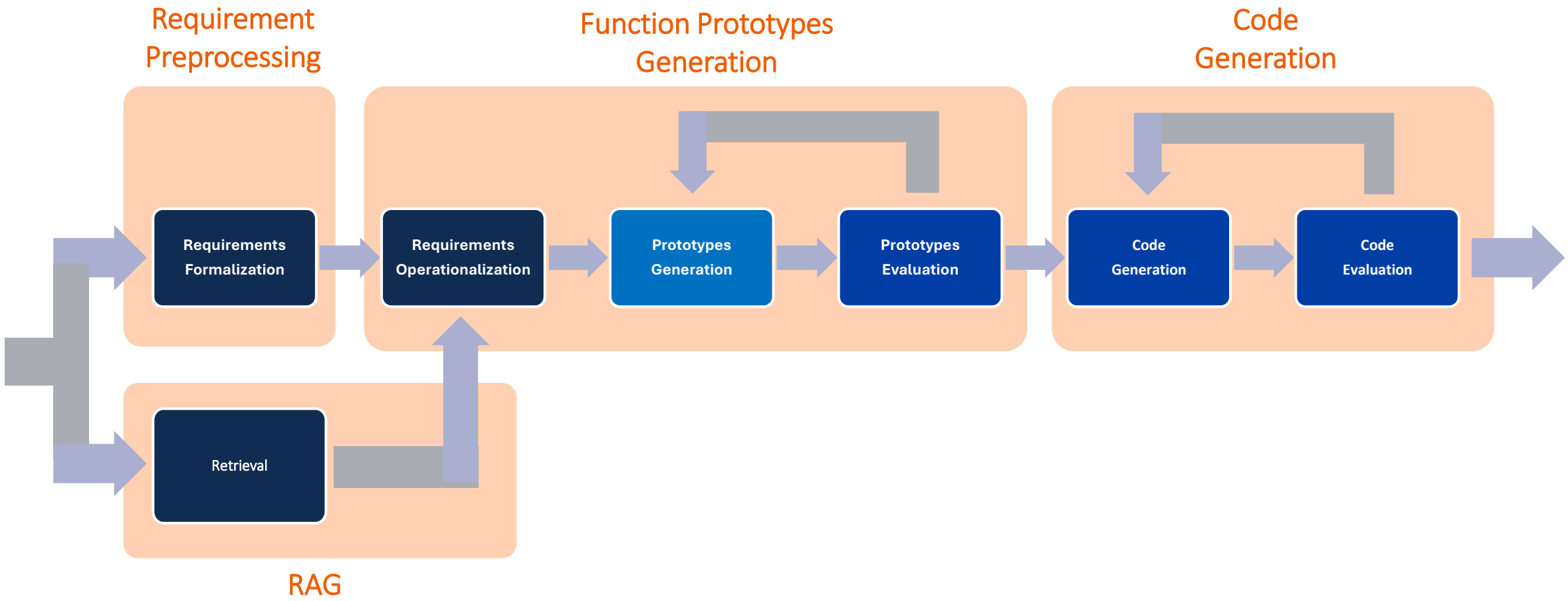
On timer expiry produce DebounceTimer_Expired event and evaluate ButtonLowThroughoutDebounce to drive subsequent actions.

Operation 2: Write_LIN_BtnCustomSts_Pressed

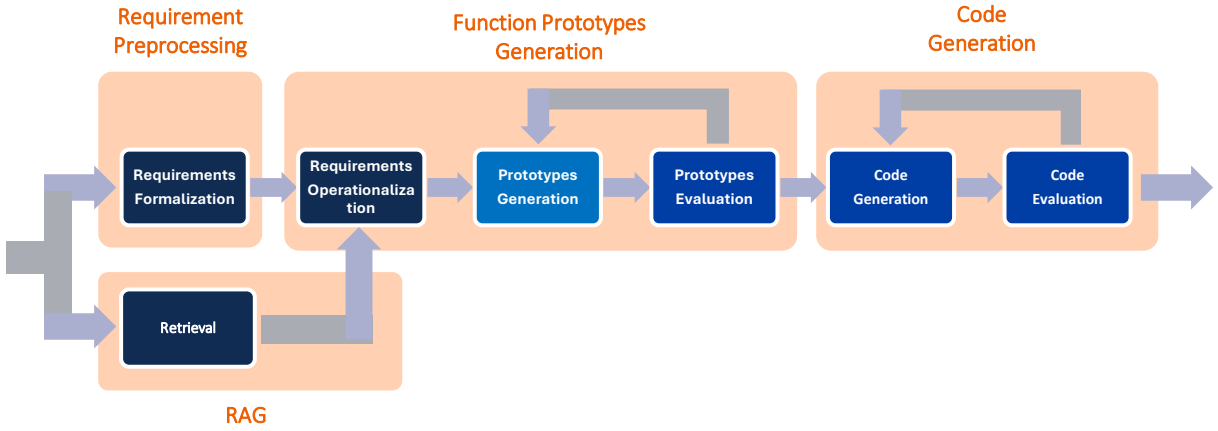
Transmit LIN signal setting LIN_BtnCustomSts to Pressed (1) when debounce completes and button has been LOW throughout.

Relevant Functions: DIN_Mgm(), DIN_Config(), LIN_PutMessage()

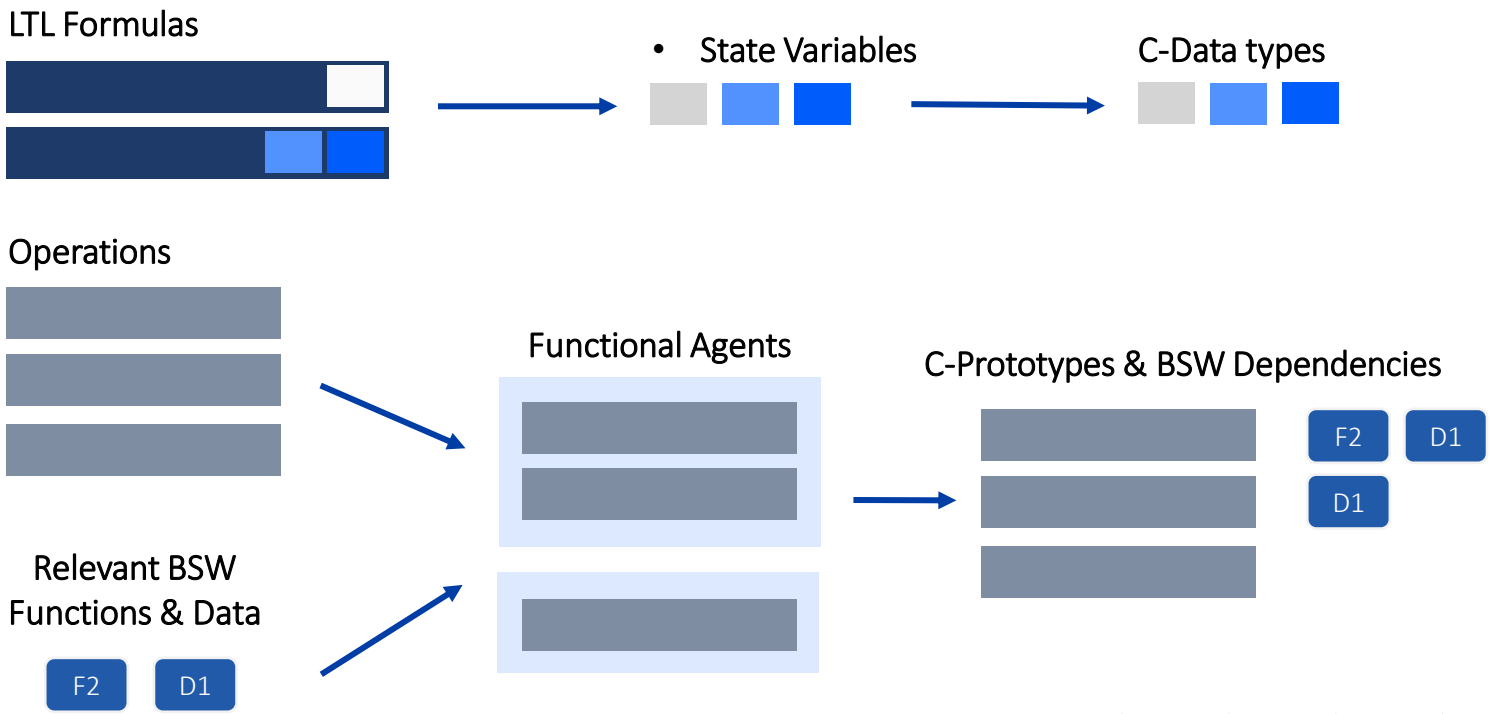
Prototypes Generation



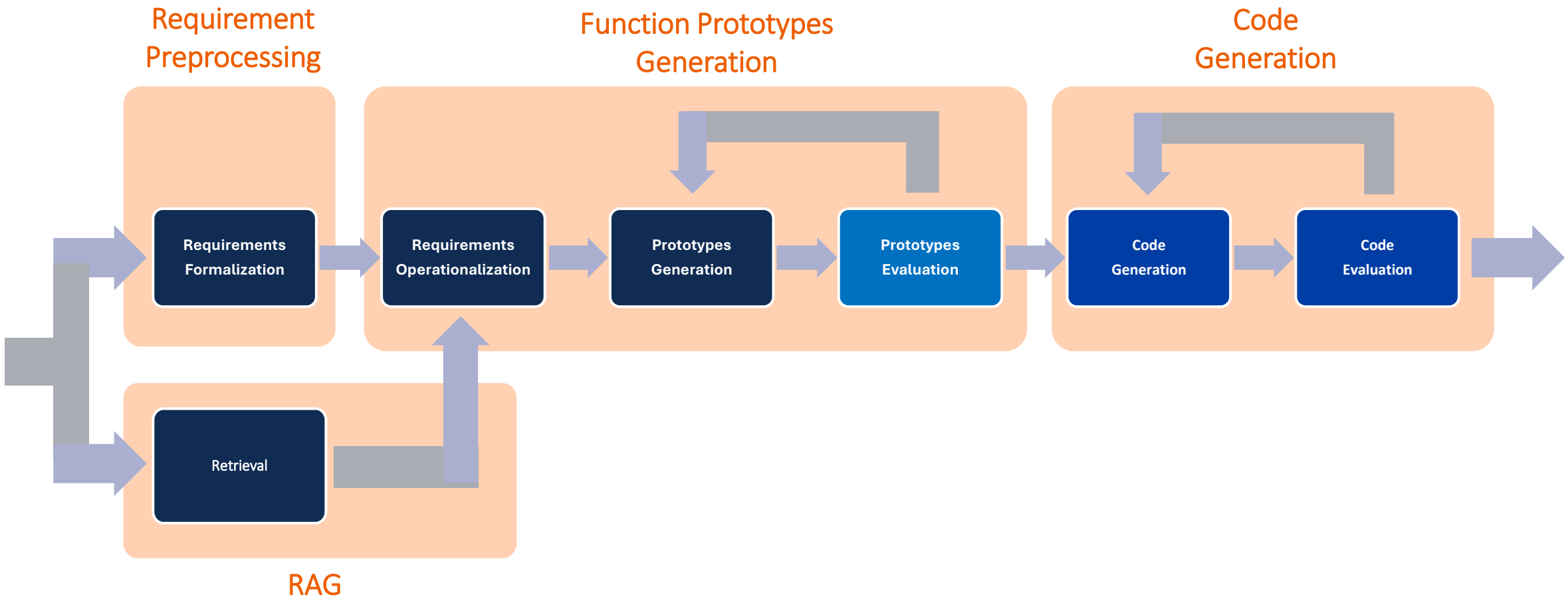
Prototypes Generation



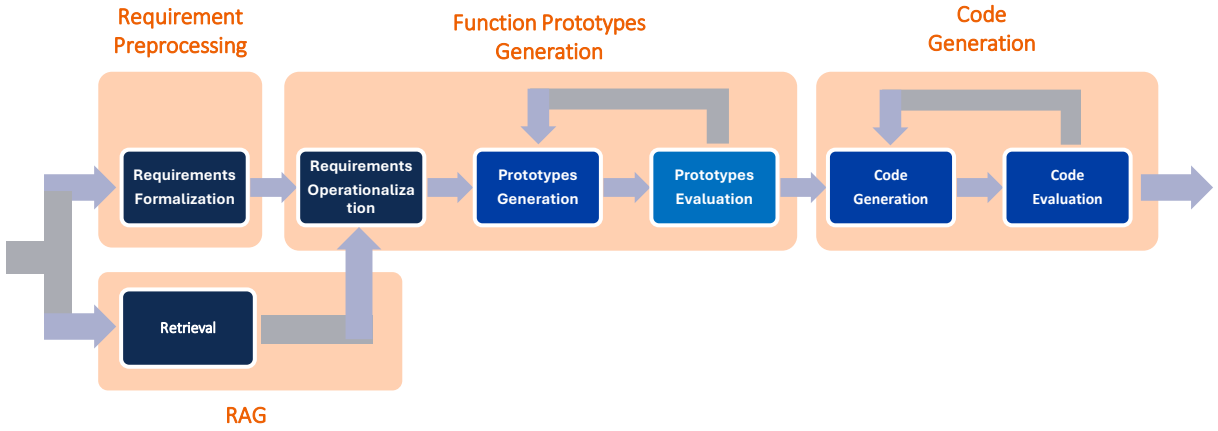
- Identify state variables from LTL formulas to **define coherent data structures**.
- Organize operations into "Functional Agents" to **define architectural roles and modules**.
- **Generate the prototypes** of the functions required to implement the identified operations.
- Map and manage **dependencies with the Basic Software** (BSW).



Prototypes Evalutation



Functional Coverage Evaluation



Evaluates whether the generated prototypes **correctly cover all the functional requirements**.

Output:

For each requirement determine a status:

{Fully Covered | Partially Covered | Not Covered}

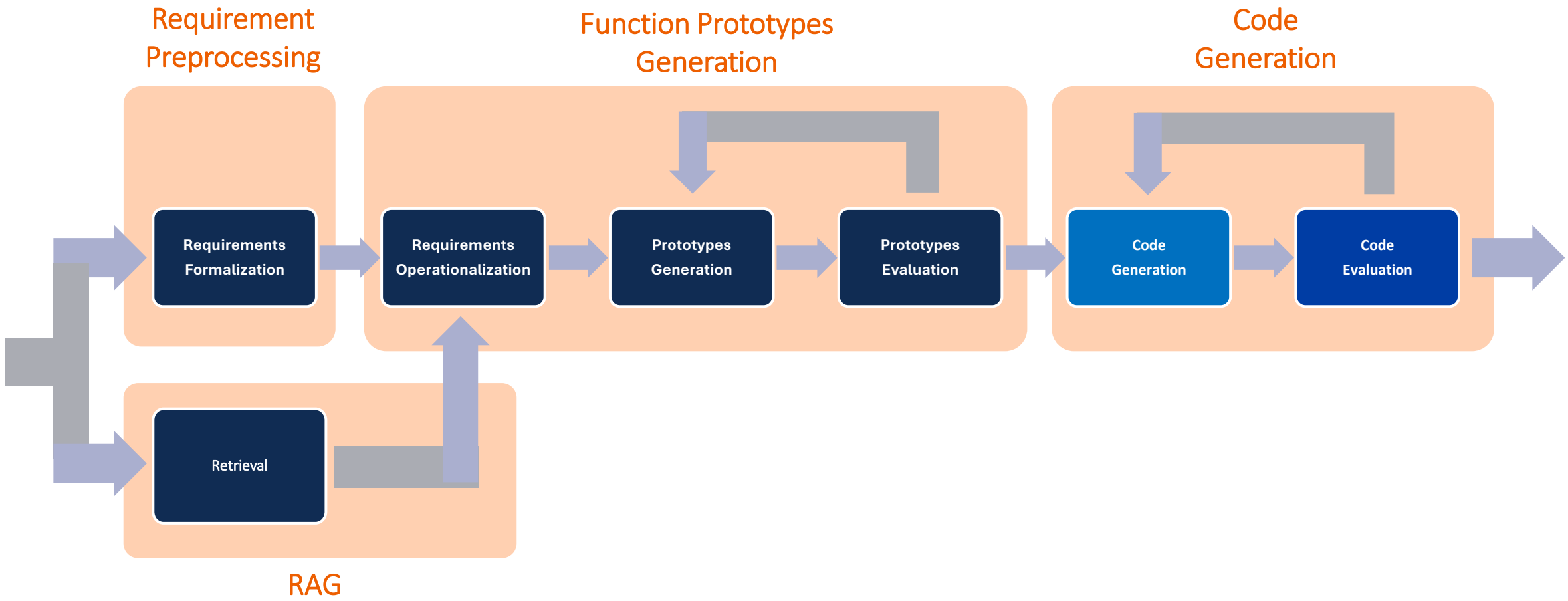
KPI:

The KPI is defined considering the corresponding scores.

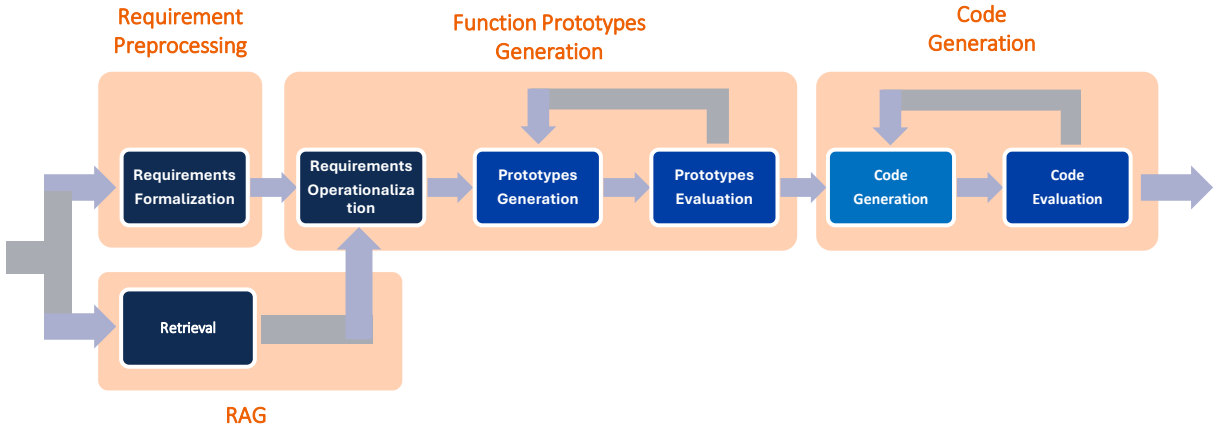
Status	Score
Fully covered	1.0
Partially covered	0.5
Not covered	0.0

$$KPI = \frac{\sum_{i=1}^{N} Score_i}{number\ of\ requirements}$$

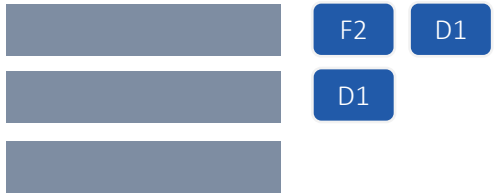
Code Generation



Code Generation



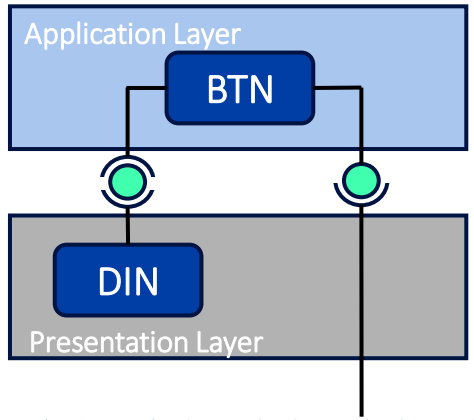
C-Prototypes & BSW Dependencies



C-Data types

LLM

```
void BTN_BtnDebounceExpiredHandler(T_DINHandle_t Chan, ...) {
    ...
    /* Query the debounced state: read-only synchronous snapshot. */
    btn_state = DIN_GetState(Chan);
    if (btn_state == S_ON)
    ...
}
```



HIS Metrics:

Comment Density: >20%

Number of Goto Statements: 0

...

MISRA C:2012

A project shall not contain unreachable code

There shall be no unused parameters in function

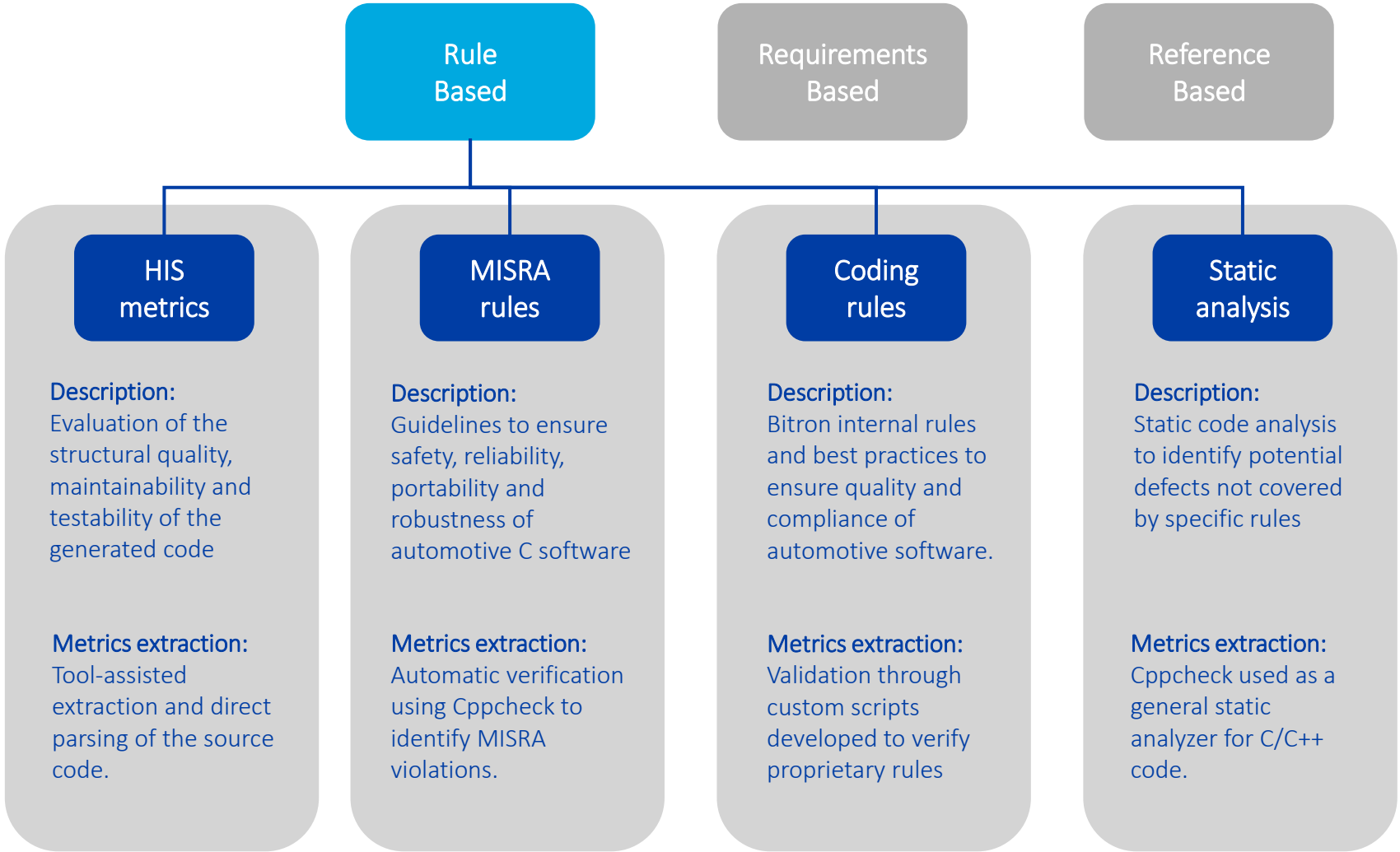
...

www.bitron.com

Evaluation Framework

Details driving Innovation

Rule-based KPI



$$KPI = \sum_{1=1}^{N} (CategoryScore_i \times Weight_i)$$

Category	Weight
HIS metrics	0.25
MISRA rules	0.35
General coding rules	0.15
Static Analysis	0.25

Requirements-based KPI

Rule Based | **Requirements Based** | Reference Based

LLM as a judge

Description
Static and rule-based techniques do not fully allow evaluating the semantics, expected behavior, and alignment with requirements of the generated code.
For this reason, an LLM-as-a-Judge framework is introduced for the structured evaluation of semantic-level and requirement-level properties of the generated embedded software

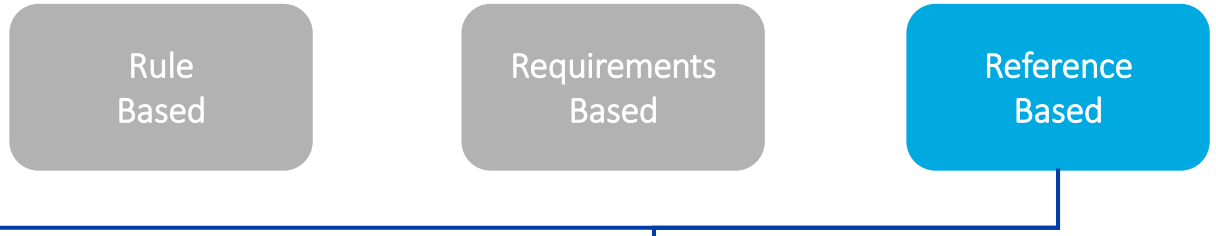
Key Elements
- Gradual evaluation of alignment with requirements
- 'Slow thinking' approach with guided reasoning
- Analysis based on categories and structured taxonomies
- Classification of inconsistencies by severity
- Final score obtained through weighted aggregation

Possible Biases
- **Illusory complexity**
  Longer or more complex code may be perceived as qualitatively better."
- **Over-trust nel reasoning**
  Convincing explanations can positively influence the model's judgment

These biases must be explicitly mitigated in the evaluation process

Adopted Techniques
- Structured prompting with defined criteria and formats
- Multi-step decomposition of the evaluation
- Simplification of the code to be analyzed
- Interpretable and traceable outputs
- Iterative refinement through structured feedback

Reference-based KPI



Deterministic

- **Entity Presence -** Verification of the presence of expected software entities (functions, structs, enums, typedefs) against the reference implementation
- **Module Presence -** Verification of the presence of expected application-layer modules
- **Basic SW Interaction -** Verification of the interactions of the generated code with the expected Basic Software (BSW) modules
- **Basic SW Callback Conformance -** Verification of the correctness of BSW callbacks:
  - missing callbacks
  - altered callbacks
  - additional callbacks
  - correctness of signature and semantics

Model - Assisted

- **Property Extraction & Classification -** Extraction and classification of properties available in the reference implementation
  - Core behaviors
  - Secondary behaviors
- **Property Matching -** Comparison between properties of the generated code and the reference implementation:
  - Fully preserved
  - Partially preserved
  - Missing
- **Additional Behaviors Identification -** Identification of additional behaviors of the generated code:
  - Beneficial or neutral
  - Questionable
  - Harmful or risky

$$KPI = \sum_{1=1}^{N} (CategoryScore_i \times Weight_i)$$

Category	Weight
Deterministic score	0.60
Model-assisted score	0.40

www.bitron.com

IT Infrastructure

Details driving Innovation

IT Infrastructure and Data Security

Enterprise architecture for the protection of industrial knowledge and AI integration

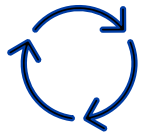
Protected Knowledge Base

- Segregated knowledge base, protected within the corporate environment
- Access and permissions managed through corporate authentication

Fundamental Guarantees

- No use of corporate data for model training
- Logical isolation between different tenants
- No data sharing with OpenAI for training
- European data residency guaranteed

Anonymized Knowledge Base



SW running in a docker container

API

LLM as service (Azure Open AI)

Integration into the Corporate Ecosystem

Integration with the same corporate Microsoft 365 tenant and with the existing IT infrastructure

Prompt Anonymization

Automatic removal of names, customer IDs, and project codes through anonymization of the knowledge base

Bitron Infrastructure

Q&A

BITRON

Details driving Innovation

Thank you

Follow us on LinkedIn

www.bitron.com