
Marmara University

Faculty Of Engineering

CSE3063

# OBJECT ORIENTED SOFTWARE DESIGN
# TERM PROJECT ITERATION 2
# Technical Impact Assessment Document

DATE TO : 26 December 2025 Submitted to: Büşra Arabacı


| | Dept | Student Id | Name Surname |
|---|---|---|---|
| 1 | CSE | 150122027 | Şeyma Özek |
| 2 | CSE | 150122060 | Fatma Zeynep Kök |
| 3 | CSE | 150123069 | Beyza Çoban |
| 4 | CSE | 150122007 | Beyza Parmak |
| 5 | CSE | 150123081 | Sultan Kocagöz |

# Table Of Contents

## 1. Executive Summary & Context:

This project started as a Python version of a Java-based chatbot system. Iteration 1 focused on basic functionality like keyword search and rule-based intent detection. Iteration 2 made the chatbot smarter and faster with features like semantic search, query splitting, and caching.

## 2. Technical Background:

**Iteration 1**

The first iteration was a direct java chatbot. It used simple keyword matching for retrieval and rule-based intent detection. While functional, the system had several limitations:

-Limited Retrieval Capability: Only supported keyword-based matching, which struggled with synonyms or nuanced queries.

-Rigid Query Handling: Could not process complex or multi-part questions effectively.

-No Optimization: Each query was processed from scratch, leading to slower performance.

**Iteration 2**

The second iteration introduced significant technical advancements that addressed these limitations and added new capabilities:

**- Semantic Search:**

- Introduced vector-based retrieval using embeddings, enabling the system to understand the meaning behind queries rather than relying solely on exact keyword matches.
- This improvement allows the chatbot to handle synonyms, paraphrased questions, and contextually similar queries more effectively.

**- Query Splitting:**

- Implemented a query decomposition mechanism to break down complex, multi-faceted questions into smaller, manageable sub-queries.
- This ensures that the chatbot can address each part of a question comprehensively, improving the accuracy of responses.

**- Caching:**

- Added a caching layer to store results of frequently asked queries, significantly reducing response times for repeated questions.
- This optimization improves user experience by making the system faster and more efficient.

**What These Changes Bring**

The advancements in Iteration 2 have transformed the chatbot into a smarter, more adaptable system. Key benefits include:

- **Improved Accuracy:** Semantic search and query splitting ensure that the chatbot provides more relevant and precise answers.
- **Enhanced Performance:** Caching reduces redundant computations, making the system faster and more responsive.

- **Scalability:** The modular design of Iteration 2 allows for easier integration of future features, such as advanced embeddings or additional retrieval strategies.
- **User Satisfaction:** By handling complex queries and providing quicker responses, the chatbot delivers a better overall user experience.

## 3. Proposed Solution Overview:

In the second iteration, the proposed solution focuses on extending the existing system architecture to support a more robust and extensible Retrieval-Augmented Generation (RAG) pipeline. Building on the modular design established in the first iteration, the system continues to treat the core logic as a black box from the user's perspective, while internally introducing additional components to improve retrieval efficiency, scalability, and evaluation capabilities.

The updated solution integrates structured document chunking, a retriever mechanism, optional reranking, and a query-level caching strategy. When a user submits a question, the system first checks whether a cached response exists. In the case of a cache hit, the answer is returned immediately, reducing latency and computational cost. For cache misses, the system executes the full retrieval pipeline, selecting relevant document chunks and generating an answer based on retrieved context. This design allows the system to balance performance and accuracy while maintaining a clean separation of concerns between components.

Additionally, Iteration-2 introduces batch processing support to enable offline evaluation of multiple queries and to collect metrics such as accuracy and average latency. Although the integration of an external large language model API was initially planned, it was excluded from the implementation scope due to time constraints. Nevertheless, the overall architecture is designed to remain LLM-agnostic, ensuring that such integrations can be incorporated in future iterations without requiring significant architectural changes.

## 4. Detailed Impact Analysis:

### 4.1. Affected Components

- **Programming Language & Runtime Environment:** The project has been fully migrated from the Java ecosystem used in Iteration 1 to the Python ecosystem.
- **Indexing & Preprocessing:** A VectorIndex (stub) providing deterministic vector simulation has been added alongside the existing KeywordIndex structure.
- **Reranking Strategy Layer:** Instead of the simple heuristic algorithm from Iteration 1, a "Hybrid Reranking" strategy has been integrated. This version supports alternative approaches such as Cosine similarity and is toggleable via the configuration file.

- **Sequential Pipeline Logic (QueryCache Integration):** The sequential execution flow (Load Config -> QueryCache Check -> Detect Intent -> Query Write -> Retrieve -> Rerank -> Answer -> QueryCache Update), managed by the RagOrchestrator component and executed via the Template Method pattern, has been updated to include the QueryCache mechanism to optimize latency .
- **Batch Execution Engine:** A new execution layer has been added to the system to automatically process a list of questions (--batch) and produce collective results, in addition to single CLI queries.
- **Evaluation Framework (EvalHarness):** An independent evaluation module has been integrated to measure system performance by calculating coverage@k, accuracy, and latency metrics.

### 4.2. Unchanged Components

Consistent with project requirements and object-oriented design principles, the following structures remain preserved:

- **Core Domain Entities:** The responsibilities and conceptual relationships of core domain entities, such as Chunk, Hit, TraceEvent, and Context, remain unchanged.
- **Fundamental OO Principles:** The modular and extensible architectural approach based on GRASP (Controller, Creator, Information Expert) and SOLID (SRP, OCP, DIP) principles has been sustained.
- **Persistence Constraints:** The constraints of not utilizing a database or Graphical User Interface (GUI) remain in effect; all data storage and configuration continue to be managed through CLI and JSON/YAML files.
- **Normalization Rules:** Basic text preprocessing rules, such as lowercasing and punctuation removal, remain valid.
- **Response Generation (AnswerAgent):** Schema validation and citation verification steps that confirm the accuracy of generated answers have maintained their place in the current structure.
- **Tracing & Logging Standards:** The standards for tracking each process step via the TraceBus and recording all events deterministically in JSONL format remain unchanged.

## 5. Out of Scope:

**Gemini API and PolicyRouter Integration:** Due to time constraints within the iteration schedule, the implementation phases for the Gemini API and PolicyRouter were excluded from the scope.