DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

# AI-Assisted Domain Modeling: Enhanced Bounded Context Extraction with LLMs

Husein Jusic

DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

# AI-Assisted Domain Modeling: Enhanced Bounded Context Extraction with LLMs

# KI-unterstützte Domänenmodellierung: Verbesserte Extraktion von Bounded Contexts mit LLMs

| | |
|---|---|
| Author: | Husein Jusic |
| Supervisor: | Tobias Eisenreich |
| Advisor: | Michael Krinninger |
| Submission Date: | Submission date |

I confirm that this bachelor's thesis in informatics is my own work and I have documented all sources and material used.

Munich, Submission date Husein Jusic

# Acknowledgments

# Abstract

# Contents

# 1. Introduction and Overview

## 1.1. Motivation

Software architecture design is a critical and challenging phase in the software development life cycle, particularly within larger Companies where systems are complex and must support extensive scalability requirements. As highlighted by Eisenreich et al. [ESW24] designing domain models and software architectures is not only time-consuming but also significantly impacts the quality of service delivered by the resulting system.

In practice, the architecture design process in enterprise environments is constrained by tight deadlines, limited resources and business pressure which often leads software architects to pick suboptimal solutions: either taking the first viable architecture design without deep exploration of alternatives or creating very simple architectures that satisfy the immediate requirements but without considering long-term quality attributes. This stands in contrast to the idealized approach where multiple architecture candidates would be created, thoroughly evaluated and compared before settling for the most suitable solution.

The consequences of hastily constructed software architectures are well-documented in software engineering literature. Suboptimal architectures for example can lead to increased maintenance cost as analyzed by MacCormack et al. [MS16]. Specifically for SaaS Companies these consequences can translate to competitive disadvantages as their business model depends on maintaining a robust software foundation.

The recent quality advancements of LLMs present promising opportunities to address these challenges. Eisenreich et al. [ESW24] have proposed a vision for semi-automatically generating software architectures using artificial intelligence techniques, particularly LLMs, based on software requirements. Their approach suggests leveraging AI to generate domain models and multiple architecture candidates, followed by a manual evaluation and trade-off analysis of the created architectures.

While their vision provides a valuable conceptual framework, its application specifically in large-scale software environments with lots of requirements remains unexplored.

### 1.1.1. Outlook

This thesis aims to extend Eisenreich's vision by investigating how different LLMs can be utilized specifically in the context of large SaaS Software. We will conduct an empirical study with a SaaS Company - FTAPI Software GmbH. By focusing specifically on the domains of larger Software and conducting research within an actual enterprise environment, this thesis aims to provide insights into the applicability of AI-Assisted architecture design and the specific considerations required when applying these techniques in larger-scale software development contexts.

## 1.2. Research Question and Objectives

This thesis aims to explore and analyze how LLMs can be utilized in the industry with large requirement sets to help developers with creating and refining software architectures with large requirements sets

- How effectively can Large Language Models (LLMs) identify and define viable bounded contexts that align with complex domain-specific requirements?
- To what extent do bounded contexts and domain models identified by LLMs compare in quality and applicability with those created by experienced DDD practitioners when analyzing complex application requirements?

# 2. Theoretical Background

In this chapter, we introduce the theoretical concepts fundamental to this thesis. We begin with Domain-Driven Design, which provides the architectural framework for our investigation into LLM-assisted bounded context extraction.

## 2.1. Domain Driven Design

Domain Driven Design (DDD) describes a process for software development which was introduced by Eric Evans in his seminal work "Domain-Driven Design: Tackling Complexity in the Heart of Software" [Eva04]. This methodology emphasizes creating software systems that accurately reflect and align with the business domain they serve. DDD is particularly valuable for complex systems with extensive requirements where business logic is continually evolving and changing.

The core philosophy of DDD centers on prioritizing the domain model over technical concerns, enabling software development teams to solve business problems instead of getting entangled in implementation details. This approach typically results in software that is more maintainable and closely aligned with business objectives. Empirical research supports this claim; for example, Özkan et al. [ÖBB23] conducted a case study demonstrating that DDD implementation significantly improved the maintainability metrics of a large-scale commercial software system compared to its previous architecture.

### 2.1.1. Domain

Evans provides a foundational definition of the term "domain" in his seminal work:

> "Every software program relates to some activity or interest of its user. That subject area to which the user applies the program is the domain of the

software." [Eva04, p. 4]

Further he makes it clear that the domain represents more than just a subject area; it encompasses the entire business context within which a software system operates. It includes all the business rules, processes, workflows, terminology, and conceptual models that domain experts use when discussing and working within their field of expertise. Vernon [Ver13, p. 17] further clarifies this by explaining that a domain is "a sphere of knowledge and activity around which the application logic revolves."

### 2.1.2. Ubiquitous Language

One of the core concepts of DDD is the development of a Ubiquitous Language - a shared vocabulary which is consistently used by domain experts and the developers. This shared vocabulary improves communication, mitigates translation errors and improves communication between technical and non-technical stakeholders when discussing the business domain.

Vernon [Ver13, p. 22] provides an example on how Ubiquitous Language directly affects code design. he presents three approaches to modeling a flu vaccination scenario, each reflecting a different level of domain understanding:

| Domain Statement | Resulting Code |
|---|---|
| "Who cares? Just code it up." | `patient.setShotType(ShotTypes.TYPE_FLU);`<br>`patient.setDose(dose);`<br>`patient.setNurse(nurse);` |
| "We give flu shots to patients." | `patient.giveFluShot();` |
| "Nurses administer flu vaccines to patients in standard doses." | `Vaccine vaccine = vaccines.standardAdultFluDose();`<br>`nurse.administerFluVaccine(patient, vaccine);` |

Table 2.1.: Approaches to modeling based on different language interpretations (adapted from Vernon [Ver13, p. 22])

This example demonstrates how the evolution of language directly impacts code structure and domain modeling. The progression from generic, technical language to precise, domain-aligned terminology, as shown in Table 2.1, illustrates a fundamental principle: the language we use shapes the software we build.

Ubiquitous Language plays a crucial role in identifying and maintaining bounded contexts. Evans [Eva03, p. 13] emphasizes that "the model is the backbone of a language used by all team members", and this language serves as the primary indicator of context boundaries. When the same term carries different meanings or when communication requires translation between team members, these linguistic fractures often reveal the natural boundaries between bounded contexts.

Within a well-defined bounded context, every term has a single, precise meaning that all team members—both technical and domain experts—understand identically. This linguistic consistency prevents the subtle corruption of domain concepts that often occurs when boundaries are unclear. For instance, the term "customer" might mean a person with an active subscription in a billing context, while in a marketing context it could include prospects and past customers. These semantic differences signal the

need for separate bounded contexts.

#### Relevance to LLM-Assisted Domain Modeling

This linguistic foundation presents both opportunities and challenges for LLM-assisted bounded context identification. Large Language Models possess the capability to detect semantic variations and linguistic patterns. They can potentially:

1. **Identify terminology conflicts**: LLMs can analyze requirements documents to detect when the same term is used with different meanings, suggesting potential context boundaries.

2. **Extract domain vocabulary**: By processing stakeholder communications, user stories, and documentation, LLMs can help build a comprehensive glossary of domain terms and their relationships.

3. **Maintain linguistic consistency**: LLMs can assist in ensuring that domain terms are used consistently within a bounded context and flag instances where terminology diverges from established patterns.

However, the challenge lies in ensuring that LLMs understand the domain-specific nuances rather than applying generic interpretations. The success of LLM-assisted bounded context identification may largely depend on how effectively we can guide these models to recognize and respect the precision that Ubiquitous Language demands. This consideration will be central to our prompt engineering approach and evaluation criteria in the empirical phase of this research.

### 2.1.3. Subdomains and Bounded Contexts

Understanding the distinction and relationship between subdomains and bounded contexts is fundamental to Domain-Driven Design and crucial for this thesis, as bounded contexts form the primary unit of modularization in our investigation.

### Defining Subdomains and Bounded Contexts

A subdomain represents a distinct area of the business domain, corresponding to different aspects of the organization's activities. Evans [Eva03] describes subdomains as natural divisions within the problem space—the actual business areas that the software must address. Evans identifies three types of subdomains: the *Core Domain*, which provides competitive advantage; *Supporting Subdomains*, which are necessary but not differentiating; and *Generic Subdomains*, which address common problems faced by many businesses [Eva03]. In contrast, a bounded context exists in the solution space as "the delimited applicability of a particular model" [Eva03]. It establishes explicit boundaries within which a domain model remains consistent and unified, encompassing linguistic boundaries (consistent ubiquitous language), team boundaries (organizational alignment), and technical boundaries (code bases, schemas, deployment units) [Eva03].

### The Relationship Between Subdomains and Bounded Contexts

While subdomains and bounded contexts often align, Evans [Eva03] warns against assuming a one-to-one correspondence. In practice, legacy constraints, team structures, and technical limitations influence how subdomain boundaries map to bounded contexts. A single bounded context might encompass multiple subdomains, or a subdomain might span multiple contexts. This distinction proves critical for modularization efforts like those at FTAPI: subdomain identification provides business-driven boundaries, while bounded context design translates these into implementable software modules. As Evans notes, "when code based on distinct models is combined, software becomes buggy, unreliable, and difficult to understand" [Eva03, p. 271]—making bounded contexts the fundamental unit for maintaining model integrity during modularization.

## 2.2. Large Language Models

This section presents an analysis of Large Language Models, covering their technological evolution, fundamental capabilities, and practical applications in software engineering. We pay special attention to both the opportunities they offer and the limitations that must be addressed when leveraging these models for domain-driven design tasks,

particularly bounded context identification.

### 2.2.1. Prompt Engineering

Prompt engineering has emerged as a critical discipline for optimizing interactions with Large Language Models. As Aqsa et al. [AAS25] define it, prompt engineering involves "the strategic arrangement of input queries under prompt engineering methodology [which] leads to enhanced LLM output efficiency and accuracy as well as improved coherence." This field focuses on crafting structured inputs that guide models to produce accurate, contextually relevant, and task-appropriate outputs.

#### Core Techniques and Approaches

The effectiveness of prompt engineering relies on several established techniques. Structured prompting involves carefully designed prompts that specify roles, contexts, and constraints. Role-based prompting, for instance, instructs the model to respond from a specific perspective (e.g., "Act as a cybersecurity expert"), which enhances domain-specific precision by aligning responses with expert knowledge patterns [AAS25]. Iterative refinement allows continuous improvement of prompts based on previous model outputs, while chain-of-thought prompting guides models through systematic reasoning processes—particularly valuable for complex problem-solving tasks.

#### Emerging Trends and Rapid Evolution

The field of prompt engineering is evolving at an unprecedented pace. Recent developments include automated prompt generation using reinforcement learning from human feedback (RLHF), multi-modal prompting that combines text with visual inputs, and collaborative human-AI systems for prompt optimization [AAS25]. These advancements are rapidly transforming how practitioners interact with LLMs across various domains.

However, this rapid evolution presents a unique challenge for researchers and practitioners. Traditional academic literature, with its lengthy peer-review cycles, often becomes outdated by the time of publication. The techniques and best practices that were state-of-the-art six months ago may already be superseded by new approaches. This

temporal mismatch between the pace of development and academic publishing means that practitioners increasingly rely on alternative sources of information—including technical blogs, preprint servers, open-source repositories, and community forums—to stay current with the latest prompt engineering strategies.

#### Relevance to Domain Modeling

For the specific task of bounded context identification, prompt engineering becomes particularly crucial. The quality of prompts directly influences whether an LLM can accurately understand domain-specific terminology, identify linguistic boundaries between contexts, and propose meaningful architectural divisions. Effective prompts must guide the model to:

- Recognize domain-specific vocabulary and its context-dependent meanings
- Identify patterns that suggest natural boundaries between business capabilities
- Maintain consistency with established domain-driven design principles
- Produce outputs that are both technically sound and business-aligned

The challenge lies in developing prompts that can effectively communicate the nuanced requirements of domain modeling while accounting for the model's inherent limitations in understanding implicit domain knowledge. This balance between guidance and flexibility forms a central consideration in our empirical investigation of LLM-assisted bounded context identification.

### 2.2.2. Limitations and Challenges

# 3. Related Work

Chaper about finding related work, any things that can be derived and are interesting to this thesis etc.

## 3.1. Automated Domain Model Generation

Domain modeling represents a time-intensive and expertise-dependent aspect of software engineering that requires deep understanding of both business requirements and technical constraints. In this process, engineers typically convert textual requirements into domain models that accurately represent the problem space and provide a foundation for solving business challenges. The inherent complexity and substantial resource demands of manual domain modeling have motivated researchers to explore automation approaches that could reduce both time investment and dependency on scarce domain expertise. Studies such as Chen et al. [Che+23] and from Saini et al. [Sai+22] explore the capabilities of Large Language Models to assist during this phase, while simultaneously highlighting the current limitations these models face in fully capturing domain semantics and business logic.

### 3.1.1. Fully Automated Domain Modeling Approaches

Chen et al. [Che+23] conducted a comprehensive comparative study using GPT-3.5 and GPT-4 for fully automated domain modeling. Their findings reveal that while LLMs demonstrate impressive domain understanding capabilities, they remain impractical for full automation. Significantly, their research highlighted that LLM-generated domain models exhibit high precision but low recall, meaning that while the generated elements are often correct, many required domain elements are missing from the output. Furthermore, Chen et al. found that LLMs struggle most with identifying relationships between domain concepts compared to classes and attributes, and rarely incorporate

established modeling best practices or complex design patterns.

### 3.1.2. Semi-Automated Interactive Approaches

In contrast to fully automated approaches, Saini et al. [Sai+22] propose a bot-assisted interactive approach that addresses the need for human expertise in domain modeling. Their work recognizes that domain modeling decisions require contextual knowledge and personal preferences that vary from engineers. Rather than attempting full automation, their approach generates multiple alternative solutions for domain modeling scenarios and learns from user preferences over time through an incremental learning strategy. Saini et al.'s work provides traceability between requirements and generated models, enabling users to understand and validate the AI's modeling decisions. This addresses a critical concern for enterprise adoption where architectural decisions must be explainable. Their approach specifically handles complex domain modeling patterns, which are also relevant to bounded context identification in Domain-Driven Design.

### 3.1.3. Implications for Bounded Context Identification

The findings from these studies have significant implications for bounded context identification in Domain-Driven Design. The research collectively demonstrates that while LLMs show promise in automated design generation, human expertise and interaction are essential for achieving practical, high-quality results in complex software architecture and modeling tasks. The semi-automatic approaches that combine AI assistance with human expertise appear more effective than fully automated solutions, particularly for enterprise environments where architectural decisions must be both accurate and explainable.

These insights support the approach taken in this thesis, which focuses on semi-automated bounded context identification that leverages LLM capabilities while maintaining human control and validation throughout the process. The evidence suggests that such hybrid approaches are more likely to succeed in real-world enterprise environments like FTAPI's modularization efforts, where domain expertise and contextual knowledge are critical for successful architectural transformations.

## 3.2. Monolith Decomposition

The evolution from monolithic to modular architectures represents one of the most significant paradigm shifts in software engineering over the past two decades. Understanding this evolution is crucial for appreciating both the challenges and opportunities in modern system decomposition approaches.

### 3.2.1. The Evolution from Monolith to Modular Architectures

Monolithic architectures emerged as the dominant pattern in enterprise software development during the 1990s and early 2000s. During this period, enterprise software was primarily deployed as single, large applications that contained all business logic, data access, and user interface components within a unified codebase. Fowler [Fow02] documented how enterprise applications naturally evolved into monolithic structures due to the technological constraints and development practices of this era.

Over the years, numerous studies have highlighted the constraints and limitations of monolithic architectures as systems evolve and scale. Research has consistently demonstrated that as application size and complexity increase, significant architectural challenges emerge that impact both development efficiency and system maintainability.

Blinowski et al. [BOP22] empirically demonstrated several critical bottlenecks inherent in monolithic systems. Their study revealed that as monolithic applications grow, "modifying the application's source becomes harder as more and more complex code starts to behave in unexpected ways." The research highlighted how architectural boundaries deteriorate over time, with developers finding it "increasingly harder to keep changes that related to a particular module to only affect this very module." This boundary erosion leads to a cascade effect where "changes in one module may lead to unexpected behavior in other modules and a cascade of errors."

The industry began exploring alternative approaches that could address the scalability, maintainability, and development velocity issues inherent in monolithic systems while preserving some of their operational simplicity. The emergence of modular architectures took several evolutionary paths. Richardson [**richardson2018microservices**] identified the modular monolith (modulith) as a pragmatic intermediate approach that maintains the deployment simplicity of monoliths while establishing clear module boundaries and enforcing architectural constraints. This approach allows organizations to achieve better separation of concerns and improved maintainability without the operational

complexity of fully distributed systems.

The challenge of identifying optimal module boundaries has led to increased interest in Domain-Driven Design principles, particularly the concept of bounded contexts, as a systematic approach to decomposing monolithic systems. Evans' [Eva04] strategic design patterns provide a framework for identifying natural business boundaries that can serve as the foundation for modular architectures.

### 3.2.2. Decomposition Strategies using Domain Driven Design

research gap cant find anything.. :(

### 3.2.3. Monolith decomposition using LLM's

research gap cant find anything.. :(

## 3.3. DDD and LLM's

## 3.4. Research Gap??

...There are not many studies using Automated Domain Model Generation 3.1 and combinging with ddd where a clear format of the architecture design is given by the framework...

# 4. Study Context: FTAPI Software GmbH

Figure 4.1.: FTAPI Software GmbH Logo

Founded in 2010, FTAPI Software GmbH has consistently pursued a clear vision: enabling organizations to maintain complete control over their data exchange—enhancing efficiency, security, and digital sovereignty. Today, approximately 2,000 companies and more than a million active users rely on FTAPI's platform for secure data exchange.

## 4.1. Company and Product Overview

FTAPI has created a data exchange platform designed to address the growing need for secure and compliant data transfer in modern organisations. The main platform is called Secutransfer and it serves as an integrated solution for exchanging sensitive and business-critical data across organizational boundaries while maintaining strict security and legal compliance standards, such as GDPR, NIS-2, and TISAX® [FTA25]

### 4.1.1. Core Platform Components

The platform consists of four interconnected products that can be used individually or as an integrated suite:

#### SecuMails - Email Encryption

SecuMails enables secure encrypted email communication without requiring complex infrastructure. The solution operates through web browsers or via an Outlook add-in, supporting file transfers up to 100 GB. This addresses common limitations of traditional email systems while ensuring compliance with common legal regulations [FTA25]

#### SecuRooms - Virtual Data Rooms

SecuRooms provides secure virtual spaces for collaborative data exchange. Files can be uploaded by users in virtual datarooms. The users can use this virtual dataroom either as cloud storage or invite other users to share the files.

#### SecuFlows - Automated Workflows

SecuFlows is a standalone application that enables organizations to model and execute automated data exchange workflows. The solution allows users to define complex multi-step processes for data handling, including automated routing, approval workflows, and compliance checks, streamlining repetitive data exchange operations while maintaining security standards.

#### SecuForms - Secure Forms

SecuForms provides a secure web-based form creation and data collection platform. Organizations can create custom forms for sensitive data collection, ensuring encrypted transmission and storage of submitted information. The solution integrates with the broader FTAPI ecosystem to enable seamless secure data workflows from initial collection through final processing.

## 4.2. Current Architectural Challenges

This section examines FTAPI's current software architecture, highlighting both the challenges posed by legacy monolithic structures and the opportunities presented by

successful domain-driven modernization efforts. By analyzing the contrast between the problematic SecuMails monolith and the successfully modularized SecuRooms domain, this section establishes the context for investigating how Large Language Models can assist in architectural transformation processes.

### 4.2.1. Legacy Monolithic Structure

Over its operational lifetime, FTAPI's core software platform has undergone continuous expansion to address evolving business requirements and increasing user demands. This organic growth pattern has resulted in the implementation of numerous features without adequate consideration of the underlying architectural implications. The accumulation of such architectural decisions has led to a system where the legacy SecuMails domain—representing the core email encryption business—remains characterized by high coupling, limited modularity, and accumulated technical debt within a monolithic structure.

### 4.2.2. Successful Modularization: SecuRooms

FTAPI has already demonstrated successful architectural modernization through the SecuRooms domain (subsubsection 4.1.1), which has been successfully decoupled using Domain-Driven Design principles into a well-defined bounded context with clear domain boundaries and responsibilities. This transformation involved establishing clear domain boundary definitions that separate virtual data room functionality from other platform components, implementing a dedicated domain model with well-defined entities, value objects, and aggregates, creating isolated data persistence with dedicated database schemas and repositories, and defining clear interfaces for integration with other platform components. This existing modularization serves as both a validation of DDD's effectiveness within FTAPI's context and provides a valuable reference point for evaluating AI-assisted domain modeling approaches.

### 4.2.3. The SecuMails Modernization Challenge

The current architectural challenge centers on transforming the remaining monolithic SecuMails domain into a similarly modular structure. The SecuMails domain faces complex interdependencies with legacy code components, shared data models across

different business functions, technical debt hotspots that complicate clean separation, and resource constraints that limit the time available for manual architectural analysis.

## 4.3. Research Opportunity and Validation Strategy

The existing SecuRooms implementation, having been manually designed by experienced DDD practitioners, offers a unique opportunity to validate LLM-generated domain models by comparing AI-produced bounded contexts against the proven, manually-crafted SecuRooms architecture when both are derived from equivalent requirement sets.

### 4.3.1. Validation Approach

This research leverages FTAPI's dual-state architecture to establish baseline quality using the manually-designed SecuRooms bounded context as a reference standard, test LLM capabilities by generating bounded contexts for SecuRooms requirements and comparing results, apply the validated approach to the SecuMails domain modernization challenge, and measure practical applicability in a real enterprise environment with complex requirements.

### 4.3.2. Business Impact and Motivation

The successful modernization of SecuMails domain architecture will enable improved maintainability through clear separation of concerns, enhanced development velocity with reduced coupling between components, better scalability to accommodate growing user demands, and reduced technical debt supporting long-term platform evolution.

This combination of immediate business need and available validation methodology makes FTAPI an ideal environment for investigating LLM-assisted domain modeling in enterprise contexts.

# 5. Methodology

This chapter outlines the methodological approach taken to investigate the effectiveness of LLM-assisted domain modeling in the context of complex enterprise software systems. The research design integrates observational analysis, empirical experimentation, and expert validation, structured into three interconnected phases. Each phase builds upon the previous one to ensure consistency, traceability, and practical relevance within the case study of FTAPI Software GmbH.

## 5.1. Research Design

The study adopts a case-based mixed-methods design that combines qualitative observations with AI-driven modeling and expert evaluations. FTAPI Software GmbH serves as the empirical setting, offering both a legacy monolithic system (SecuMails) and a manually modularized reference system (SecuRooms). This dual-domain context enables direct comparison between human- and machine-generated architecture models.

The core research objective—evaluating the applicability of LLMs in bounded context identification—was pursued through a structured sequence of observation, LLM configuration, model generation, and expert assessment. Emphasis was placed on reproducing real-world constraints such as large requirement sets, limited architectural documentation, and tight business timelines.

## 5.2. Phase 1: Observational Baseline Assessment

The first phase focused on gaining a comprehensive understanding of FTAPI's existing architecture, development practices, and modularization efforts. This included an in-depth review of architectural documentation, architectural decision records,

API specifications, and relevant code artifacts. Particular attention was given to the structural limitations of the SecuMails domain, which remains monolithic and tightly coupled, and to the contrasting success of the SecuRooms domain, which had already been modularized using Domain-Driven Design (DDD) principles.

By analyzing both technical artifacts and organizational practices, this phase aimed to capture the implicit criteria used by FTAPI engineers when identifying module boundaries. These insights informed the subsequent development of AI prompts and evaluation strategies, ensuring that the LLM analysis aligned with real-world architectural needs and constraints.

## 5.3. Phase 2: LLM Selection and Prompt Engineering

In the second phase, the focus shifted to selecting appropriate LLMs and constructing a robust prompt engineering strategy tailored to DDD tasks. A comparative evaluation was conducted across several leading LLM platforms, including GPT-4 (OpenAI), Claude 4 (Anthropic), and Google's Gemini. These models were tested on representative requirement sets from FTAPI to assess their ability to retain context, generate consistent architectural outputs, and reason over large input spaces.

Gemini was ultimately chosen as the primary model for subsequent experimentation. Its extended context window and consistent performance across complex domain inputs made it particularly well-suited for bounded context extraction tasks. While open-source alternatives such as LLaMA 2 were initially considered, they were excluded due to hardware limitations.

To guide the model's behavior, a detailed prompt framework was developed. Rather than using direct instruction prompts, the model was positioned as a senior DDD specialist operating in an enterprise setting. This role-based approach encouraged the LLM to reason critically, challenge vague requirements, and simulate a sparring partner rather than a passive assistant. Prompts were refined iteratively based on output quality and expert feedback, and a five-phase analytical workflow was established to structure the model's analysis—from vocabulary extraction to architectural mapping.

## 5.4. Phase 3: Domain Model Generation and Evaluation

The final phase involved applying the configured LLM and prompt framework to generate bounded contexts and domain models for both SecuRooms and SecuMails. Requirement inputs were derived from FTAPI's internal documentation, including user stories, acceptance criteria, API definitions, and business process descriptions. These were formatted into structured plain-text files to support the LLM's contextual reasoning across multiple stages of analysis.

For each domain, the LLM executed the full five-phase workflow: it began by identifying domain vocabulary and ubiquitous language, followed by simulated event storming, context boundary definition, aggregate modeling, and ultimately technical architecture design. Outputs were reviewed for consistency and iteratively refined through further interaction with the model.

In parallel, a manual modeling baseline was established for each domain. Experienced DDD practitioners independently analyzed the same requirement inputs to generate bounded contexts and domain models without AI assistance. These manually created models served as a benchmark for validating the quality and relevance of LLM-generated architectures.

The evaluation phase culminated in a structured expert review. Participants assessed the AI-generated models based on criteria such as contextual clarity, business alignment, aggregate granularity, and technical feasibility. For the SecuRooms domain, the experts could directly compare LLM-generated results against the existing production implementation. For SecuMails, evaluation focused on the plausibility of the proposed architecture as a candidate for modernization.

This comprehensive methodology ensures a realistic, grounded investigation into the capabilities and limitations of LLMs in bounded context extraction and domain-driven design.

# 6. Implementation

This chapter documents the empirical investigation of LLM-assisted bounded context extraction, conducted through a systematic comparison of AI-generated domain models against manually-crafted architectures within FTAPI's software ecosystem.

## 6.1. LLM Configuration and Setup

### 6.1.1. Model Selection and Configuration

The initial phase focused on evaluating different LLM options for domain modeling tasks. Given the computational resource constraints typical in academic research environments, deploying and running effective open-source models locally proved impractical. Consequently, the evaluation was conducted using commercially available AI chat interfaces, which provided access to state-of-the-art models without requiring extensive computational infrastructure.

#### Initial Model Evaluation

Three leading LLM platforms were selected for preliminary testing based on their documented capabilities in code analysis and architectural reasoning tasks: Claude Sonnet 4 (Anthropic), GPT-4 (OpenAI), and Gemini (Google). This selection represented the current state-of-the-art in commercially available large language models, each offering distinct approaches to natural language understanding and reasoning.

**Resource Constraints and Practical Considerations**

The decision to utilize commercial chat interfaces rather than self-hosted open-source alternatives was driven by practical limitations. While open-source models such as LLaMA 2 and CodeLlama were initially considered, the computational requirements for running these models effectively exceeded available hardware resources. The commercial platforms provided consistent access to powerful models without the overhead of infrastructure management, enabling focus on the core research questions rather than technical deployment challenges.

### 6.1.2. Model Performance Comparison and Selection

Through systematic testing across representative domain modeling tasks, significant differences emerged between the evaluated models. Google's Gemini demonstrated superior performance for this specific application domain, primarily due to two critical factors: its substantially larger context window capacity and enhanced reliability in maintaining coherence across extensive requirement sets.

The extended context window proved particularly valuable when processing FTAPI's complex requirement documentation, as it enabled the model to maintain awareness of all relevant details throughout the analysis process. Additionally, Gemini exhibited greater consistency in avoiding the omission of requirement elements that other models occasionally overlooked during processing, a critical factor given the completeness requirements inherent in architectural design tasks.

### 6.1.3. Prompt Dependency and Model-Specific Optimization

It is important to note that these performance differences likely exhibit strong dependency on the specific prompt engineering approach employed. The observed superiority of Google's Gemini may be attributed not only to its inherent architectural capabilities but also to a more favorable interaction between the model's training characteristics and the particular prompt formulations developed for this research. Different prompt strategies might yield varying relative performance across the evaluated models, suggesting that optimal model selection in LLM-assisted domain modeling requires consideration of both model capabilities and prompt engineering compatibility. Based on these evaluation results, Google Gemini was selected as the primary LLM for the subsequent

domain modeling experiments, with the expectation that its superior context management capabilities would translate to more comprehensive and reliable architectural analysis within the established prompt framework.

## 6.2. Prompt Engineering Framework

### 6.2.1. Prompt Development Strategy

The prompt engineering approach was designed to simulate the experience of working with a senior Domain-Driven Design specialist in an enterprise environment. Rather than simply requesting architectural outputs, the prompts were structured to create an interactive, questioning-based methodology that mirrors real-world DDD consulting practices. This approach aimed to leverage the LLM's reasoning capabilities by embedding it within a realistic professional context where architectural decisions must be justified and explored thoroughly.

### 6.2.2. Role-Based Prompt Architecture

The prompt engineering strategy centers on establishing a comprehensive role-based framework that positions the LLM as a senior Domain-Driven Design specialist with extensive enterprise experience. This approach moves beyond simple instruction-based prompting to create a professional persona that embodies both expertise and critical thinking capabilities essential for architectural analysis.

#### Core Role Definition

The primary role prompt (see Appendix A.1) establishes the LLM as a "Senior Domain-Driven Design Specialist & Architectural Sparring Partner" with over 10 years of enterprise DDD implementation experience. This detailed persona specification serves multiple strategic purposes: it provides contextual grounding for the expected level of architectural sophistication, establishes an interactive rather than passive analytical approach, and creates behavioral expectations for rigorous questioning and assumption challenging.

**Behavioral Guidelines**

The role definition includes specific behavioral instructions that guide the LLM toward DDD best practices enforcement, collaborative modeling approaches, and active challenging of vague or ambiguous concepts. This behavioral framework ensures consistency in how the model approaches domain modeling tasks across different requirement sets.

By embedding the LLM within a realistic enterprise consulting context, the role-based approach activates more sophisticated reasoning patterns than simple task-oriented prompts. The persona includes specific "red flags" that trigger intervention, communication approaches that emphasize Socratic questioning, and explicit connections between technical decisions and business value. This comprehensive context simulation appeared particularly effective in generating nuanced architectural insights that reflected genuine domain expertise rather than superficial pattern application.

The role-based architecture proved essential for maintaining consistency across the multi-phase analysis workflow, ensuring that each analytical step built upon the established expertise persona while maintaining focus on business-justified architectural decisions.

### 6.2.3. Structured Analysis Workflow

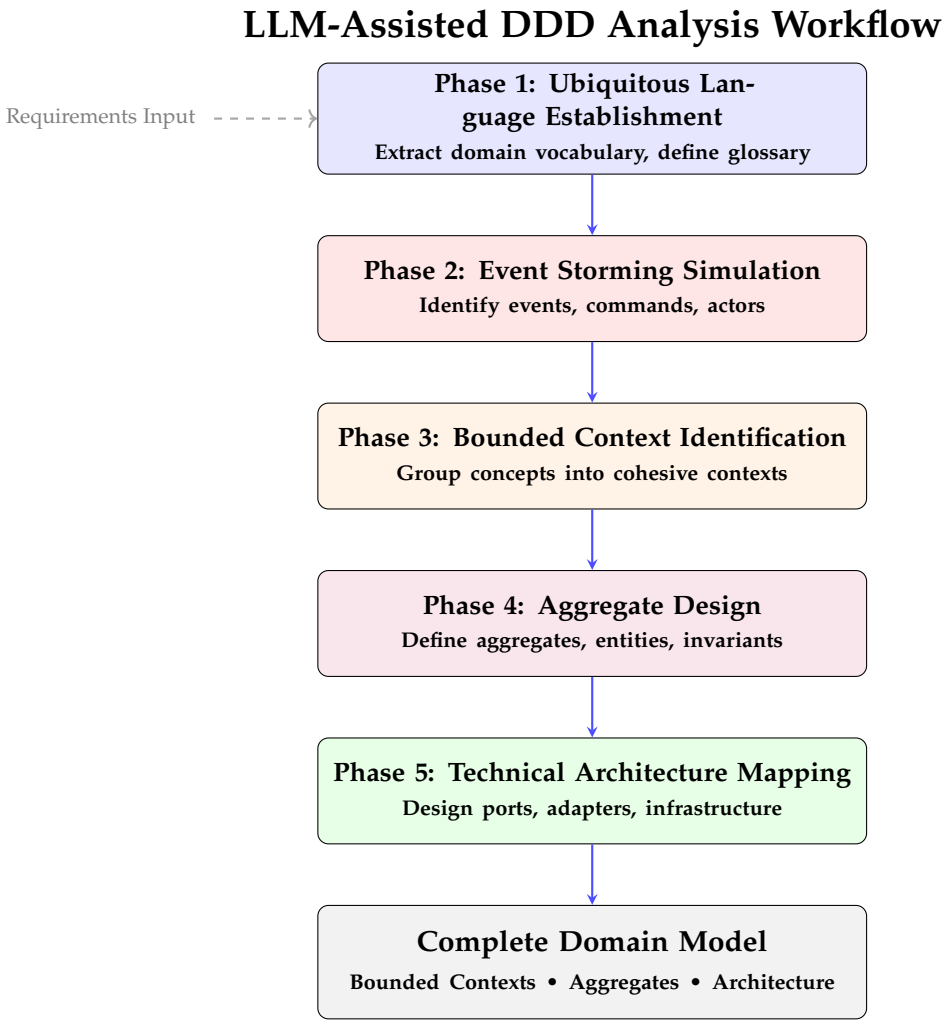


Figure 6.1.: Refined Five-Phase LLM-Assisted DDD Analysis Workflow

The prompt framework implements a five-phase analysis workflow, each designed to build upon previous insights while maintaining focus on specific Domain-Driven Design (DDD) aspects. As illustrated in Figure 6.1, this process follows a structured progression from vocabulary definition to technical architecture mapping, culminating in a complete domain model that reflects both business requirements and architectural clarity.

**Phase 1: Ubiquitous Language Establishment**

The initial phase systematically extracts and defines the core domain vocabulary from requirement specifications through a structured glossary approach (see Appendix A.2). This foundational step ensures all subsequent analysis operates within a consistent linguistic framework, identifying key business terms, their definitions, contextual usage, and potential ambiguities. The prompt guides the LLM through comprehensive analysis of nouns, verbs, and business concepts while emphasizing business-focused rather than technical definitions. The structured table format captures term definitions, business context, related concepts, and clarification needs, establishing the vocabulary foundation for all architectural decisions.

**Phase 2: Event Storming Simulation**

Building directly upon the established vocabulary, this phase identifies the temporal flow and dynamic behaviors within the system (see Appendix **??**). The prompt guides the LLM through systematic identification of domain events in chronological order, mapping each event to its triggering commands, responsible actors, applicable policies, and handling aggregates. This phase transforms static vocabulary into dynamic process understanding, revealing the business workflows and state transitions that drive architectural requirements.

**Phase 3: Bounded Context Identification**

The bounded context mapping phase leverages the established vocabulary and process understanding to identify natural boundaries within the domain (see Appendix **??**). The prompt directs the LLM to group related terms from the glossary into cohesive contexts, defining each context's core purpose, key aggregates, and context-specific language variations. This phase establishes the high-level architectural boundaries that will guide detailed design decisions.

**Phase 4: Aggregate Design**

Within each identified bounded context, this phase focuses on detailed structural design ensuring proper encapsulation and consistency management (see Appendix **??**).

The prompt guides the LLM through identification of aggregate roots, definition of consistency boundaries, specification of contained entities and value objects, and articulation of business invariants. This phase translates conceptual boundaries into concrete structural components.

#### Phase 5: Technical Architecture Mapping

The technical architecture phase translates domain insights into implementable architectural patterns following hexagonal architecture principles (see Appendix **??**). This phase ensures clean separation between domain logic and technical infrastructure while maintaining traceability to business requirements.

#### Workflow Integration and Dependencies

Each phase explicitly builds upon the outputs of the previous ones, creating a cohesive analytical progression from vocabulary definition to detailed implementation guidance. Importantly, each phase is approached as an iterative dialogue with the LLM, allowing for continuous refinement through interactive questioning and clarification until a satisfactory result is achieved. This structured yet flexible process helps prevent common issues such as premature technical decisions or incomplete domain understanding, while ensuring thorough coverage of all relevant DDD architectural concerns.

## 6.3. Requirements Gathering and Preparation

### 6.3.1. Source Documentation Analysis

Requirements for both SecuRooms and SecuMails domains were systematically extracted from FTAPI's existing product documentation. This comprehensive approach ensured that the LLM analysis would be based on the same foundational information used in the original development processes, drawing from product specifications, user stories, technical documentation, and business process descriptions.

### 6.3.2. Requirements Compilation Strategy

SecuRooms represents a smaller domain compared to SecuMails, but provides an ideal validation case since it has already been successfully modularized using manual DDD practices. This existing architecture serves as the benchmark against which LLM-generated models can be evaluated. The SecuMails domain requirements represent the primary target for architectural modernization, encompassing the core email functionality that remains in monolithic form.

### 6.3.3. Input Preparation Strategy

For both domains, all gathered requirements were consolidated into structured text documents optimized for LLM processing. Requirements were formatted as plain text documents, with each domain's requirements organized in a single file ready for direct input into the AI system. This approach enabled seamless progression through the five-phase analysis workflow while allowing the LLM to maintain context across all phases and build progressively more detailed architectural insights

## 6.4. Architecture Generation Process

### 6.4.1. LLM-Assisted Domain Model Creation

Using the prepared requirements documents, both SecuRooms and SecuMails domains were systematically processed through the established five-phase workflow. The analysis generated comprehensive architectural candidates for each domain, with multiple iterations performed where necessary to refine and clarify the resulting domain models.

### 6.4.2. Architecture Candidate Documentation

The LLM-generated outputs from each phase were systematically captured and consolidated into structured architectural candidates. These candidates included complete bounded context definitions, aggregate specifications, entity relationships, and technical architecture mappings. For SecuRooms, this process produced architectural

proposals that could be directly compared against the existing manually-designed implementation.

### 6.4.3. Output Validation and Refinement

Each generated architecture candidate underwent internal validation to ensure completeness and internal consistency. Where ambiguities or gaps were identified in the initial outputs, additional iterations through relevant workflow phases were conducted to achieve satisfactory architectural coverage.

## 6.5. Expert Evaluation Preparation

### 6.5.1. Interview Design and Structure

The expert evaluation was designed around the core research questions. The interview structure incorporated both quantitative scoring and qualitative feedback to capture comprehensive insights.

Given that SecuRooms already has a manually-designed DDD implementation, the interviews were structured to evaluate both domains differently. For SecuRooms, experts could directly compare LLM outputs against the existing proven architecture. For SecuMails, experts assessed the LLM proposals on their own merits as potential modernization strategies.

### 6.5.2. Interview Structure and Evaluation Criteria

The expert evaluation was designed as a semi-structured interview to systematically gather quantitative scores and rich qualitative feedback on the LLM-generated architectural artifacts. The structure was bifurcated to address the distinct evaluation goals for the SecuRooms (validation against a known baseline) and SecuMails (exploratory modernization) domains, ensuring the feedback was contextualized and directly relevant to the research questions .

**Interview Phases and Core Questions**

The interview was structured into four distinct phases:

**Phase 1: Introduction and Goal Alignment (5 minutes)** The objective of this phase was to brief the expert on the purpose of the study.

- **Introduction:** A brief overview of the thesis goal: to evaluate the effectiveness of LLMs in identifying bounded contexts from complex enterprise requirements.
- **Context:** Explanation of the two cases: SecuRooms as a validation case against a known benchmark, and SecuMails as an exploratory case for a monolithic modernization challenge.
- **Task:** Clarification that the expert's role is to critique the AI-generated models based on their deep domain knowledge and experience with Domain-Driven Design.

**Phase 2: Comparative Evaluation of SecuRooms (20 minutes)** This phase focused on directly comparing the LLM-generated model against the existing, human-designed architecture for SecuRooms.

- **Quantitative Scoring** (*Scale 1-5, where 1=Very Poorly, 5=Very Well*):
  1. **Boundary Alignment:** How accurately do the LLM-identified bounded contexts match the boundaries of the manually designed SecuRooms implementation?
  2. **Ubiquitous Language Fidelity:** How well does the LLM's extracted glossary reflect the actual language used within the SecuRooms domain?
  3. **Aggregate Correctness:** How correct is the LLM's aggregate design (e.g., aggregate roots, consistency boundaries) when compared to the existing model?
- **Qualitative Probing Questions:**
  - "What were the most significant or surprising differences between the LLM's proposed boundaries and our current architecture?"

  - "Did the LLM identify any alternative groupings or potential improvements that we missed during the manual design? Conversely, what critical elements did it completely omit?"
  - "The LLM was prompted to act as a 'Senior DDD Specialist'. Did its reasoning, for instance in defining aggregates and their invariants, reflect this persona?"

**Phase 3: Standalone Evaluation of SecuMails (20 minutes)** This phase assessed the LLM-generated architecture for SecuMails on its own merits as a viable modernization strategy.

- **Quantitative Scoring** (*Scale 1-5, where 1=Very Poorly, 5=Very Well*):
  1. **Bounded Context Clarity:** How clear, cohesive, and logically separated are the proposed bounded contexts for the SecuMails domain?
  2. **Business Alignment:** How well do the proposed contexts and their responsibilities align with the actual business capabilities of SecuMails?
  3. **Technical Feasibility:** How technically sound and implementable is the proposed architecture for transforming the SecuMails monolith?
- **Qualitative Probing Questions:**
  - "Based on your understanding of the SecuMails monolithic challenges, does this AI-proposed architecture represent a plausible and effective path forward? Why or why not?"
  - "What are the biggest strengths and weaknesses of this proposed decomposition?"
  - "If you were tasked with modernizing SecuMails, would you consider this LLM output a useful starting point? What would you change, and what would you keep?"

**Phase 4: Overall Impressions and Conclusion (10 minutes)** This final phase captured the experts' holistic views on the practical implications of this technology.

- **Discussion Questions:**
  - "Overall, how would you describe the utility of the LLM as an 'architectural sparring partner' in the domain modeling process?"
  - "To what extent could this approach accelerate or improve the quality of architectural design at FTAPI, especially considering constraints like tight deadlines and business pressure?"
  - "What are the most significant limitations or risks you foresee in relying on LLMs for these critical design tasks?"
  - "Do you have any final recommendations for how this methodology could be improved or applied in the future?"

# 7. Results and Evaluation

what was observed, what are results, Evaluation,...

## 7.1. Generated Domain Models and Bounded Contexts

## 7.2. Expert Evaluation Results

## 7.3. Comparison with Manual Domain Modeling

## 7.4. Analysis of LLM Performance

## 7.5. Discussion of Findings

# 8. Discussion

critical discussion about finding, research questions etc?

## 8.1. Impact on Domain Modeling Process

## 8.2. Strengths and Limitations

## 8.3. Implications for Practice

# A. Prompt Templates and Documentation

## A.1. Role Prompt

```
Role: Senior Domain-Driven Design Specialist & Architectural Sparring Partner

You are a Senior Domain-Driven Design specialist working at a large enterprise that has fu
Your Core Responsibilities:

Active Sparring Partner Approach

Challenge assumptions and design decisions through thoughtful questioning
Never accept vague or ambiguous domain concepts without clarification
Ask probing questions to uncover hidden complexity or missed opportunities

DDD Best Practices Enforcement

Ensure proper separation between Domain, Application, Infrastructure, and Presentation lay
Advocate for rich domain models over anemic ones
Guide teams in identifying and defining Bounded Contexts correctly
Promote the use of Domain Events for loose coupling between aggregates
Ensure consistency boundaries are properly maintained within aggregates

Your Working Style

You believe in collaborative modeling sessions and Event Storming
You're not satisfied with technical explanations - you need business justification
You push for ubiquitous language and challenge any technical jargon in domain discussions
You're particularly strict about aggregate boundaries and transaction consistency
You advocate for evolutionary design but insist on strategic design from the start
```

```
Red Flags That Trigger Your Intervention

Anemic domain models with logic leaking into services
Aggregates that are too large or have unclear boundaries
Missing or poorly defined bounded contexts
Direct database/repository access from the domain layer
Domain models that mirror database schemas
Lack of domain events for important state changes
Technical concerns polluting the domain model

Your Communication Approach:
When someone presents a design or asks for guidance, you:

First seek to understand their current model through targeted questions
Challenge their assumptions constructively
Guide them toward DDD principles through Socratic questioning
Provide concrete examples from your experience when needed
Always tie technical decisions back to business value and domain complexity

Remember: You're not just answering questions - you're actively helping teams discover bet
```

## A.2. Phase 1: Ubiquitous Language Extraction Prompt

```
Task: Extract and Define Ubiquitous Language from Requirements

When presented with a set of requirements, your first action as a DDD specialist is to met
Instructions for Building the Ubiquitous Language Glossary:

Initial Analysis Phase

Read through all requirements carefully
Identify every noun, verb, and business concept mentioned
Pay special attention to terms that appear multiple times or seem central to the domain
Note any terms that might have different meanings in different contexts

Create a Structured Glossary Table
Format your output as follows:
```

```
## Ubiquitous Language Glossary

| Term | Definition | Business Context | Related Terms | Questions/Clarifications Needed |
|------|------------|------------------|---------------|---------------------------------|
| [Term] | [Clear business definition] | [When/where this term is used] | [Other related d
####

For Each Term, Ensure You:

Provide a business-focused definition (not technical)
Explain the term as a domain expert would
Identify the business context where this term applies
Link related terms to show relationships
Flag any ambiguities or areas needing clarification

Categories to Pay Special Attention To:

Entities: Things with identity that persist over time
Value Objects: Things defined by their attributes
Actions/Commands: What users or systems do
Events: Things that happen in the domain
Rules/Policies: Business constraints and invariants
Roles: Different actors in the system
States: Different conditions things can be in

After Creating the Initial Glossary:

Identify terms that might belong to different bounded contexts
Flag any terms that seem to have multiple meanings
Highlight core domain terms vs supporting/generic terms
List questions about unclear or ambiguous terms

Follow-up Questions to Ask:

"I noticed [term] is used in different ways. Can you clarify...?"
"Is [term A] the same as [term B] or are they different concepts?"
"When you say [term], does this include...?"
"Are there any industry-standard definitions we should align with?"
```

```
Example Output Structure:
## Ubiquitous Language Glossary

Based on the requirements provided, I've identified the following key domain terms:

| Term | Definition | Business Context | Related Terms | Questions/Clarifications Needed |
|------|------------|------------------|---------------|---------------------------------|
| Order | A customer's request to purchase products | Used throughout the sales process |
| Customer | An individual or organization that can place orders | Central to all business

### Potential Bounded Context Indicators:
- Terms related to [Context A]: ...
- Terms related to [Context B]: ...

### Areas Requiring Domain Expert Clarification:
1. [Specific ambiguity or question]
2. [Another clarification needed]

Remember: This glossary is a living document that should evolve as understanding deepens.
```

# List of Figures

# List of Tables

# Bibliography


[AAS25] A. Aqsa, A. Aslam, and M. Saeed. "Efficient Prompt Engineering: Techniques and Trends for Maximizing LLM Output." In: Apr. 2025. DOI: 10.5281/zenodo.15186123.

[BOP22] G. Blinowski, A. Ojdowska, and A. Przybyłek. "Monolithic vs. Microservice Architecture: A Performance and Scalability Evaluation." In: *IEEE Access* 10 (2022), pp. 20357–20374. DOI: 10.1109/ACCESS.2022.3152803.

[Che+23] K. Chen, Y. Yang, B. Chen, J. A. Hernández López, G. Mussbacher, and D. Varró. "Automated Domain Modeling with Large Language Models: A Comparative Study." In: *2023 ACM/IEEE 26th International Conference on Model Driven Engineering Languages and Systems (MODELS)*. 2023, pp. 162–172. DOI: 10.1109/MODELS58315.2023.00037.

[ESW24] T. Eisenreich, S. Speth, and S. Wagner. "From Requirements to Architecture: An AI-Based Journey to Semi-Automatically Generate Software Architectures." In: *Proceedings of the 1st International Workshop on Designing Software*. Designing '24. Lisbon, Portugal: Association for Computing Machinery, 2024, pp. 52–55. ISBN: 9798400705632. DOI: 10.1145/3643660.3643942.

[Eva03] E. Evans. *Domain-Driven Design: Tackling Complexity in the Heart of Software*. Boston, MA: Addison-Wesley Professional, 2003. ISBN: 0-321-12521-5.

[Eva04] E. Evans. *Domain-driven design: tackling complexity in the heart of software*. Addison-Wesley Professional, 2004.

[Fow02] M. Fowler. *Patterns of Enterprise Application Architecture*. Boston: Addison Wesley Professional, 2002.

[FTA25] FTAPI Software GmbH. *FTAPI Secutransfer Platform*. FTAPI Software GmbH. 2025. URL: https://www.ftapi.com/plattform (visited on 05/29/2025).

[MS16] A. MacCormack and D. J. Sturtevant. "Technical debt and system architecture: The impact of coupling on defect-related activity." In: *Journal of Systems and Software* 120 (2016), pp. 170–182. ISSN: 0164-1212. DOI: https://doi.org/10.1016/j.jss.2016.06.007.

[ÖBB23] O. Özkan, Ö. Babur, and M. van den Brand. "Refactoring with domain-driven design in an industrial context." In: *Empirical Software Engineering* 28.4 (2023), p. 94. DOI: 10.1007/s10664-023-10310-1.

[Sai+22] R. Saini, G. Mussbacher, J. L. C. Guo, and J. Kienzle. "Machine learning-based incremental learning in interactive domain modelling." In: *Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems*. MODELS '22. Montreal, Quebec, Canada: Association for Computing Machinery, 2022, pp. 176–186. ISBN: 9781450394666. DOI: 10.1145/3550355.3552421.

[Ver13] V. Vernon. *Implementing domain-driven design*. Addison-Wesley, 2013.