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 want to introduce some theorethical concepts which are important for this thesis

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 Subdomain and Bounded Contexts

why subdomains and what are bounded contexts etc...

2.1.3 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:

This example shows how a more precise and domain-aligned language leads to code that better captures the busines reality. The third approach produces code that correctly assigns the responsibility to the appropriate domain objects. When developers and Domain Experts share this level of precision in the requirements, the resulting software more accurately reflects the domain's true processes and relationships.

Domain Statement	Resulting Code

"Who cares? Just code it up."	`patient.setShotType(ShotTypes.TYPE_FLU);`
	`patient.setDose(dose);`
	`patient.setNurse(nurse);`

"We give flu shots to patients."	`patient.giveFluShot();`
"Nurses administer flu vaccines to patients in standard doses."	`Vaccine vaccine = vaccines.standardAdultFluDose();` `nurse.administerFluVaccine(patient, vaccine);`

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

2.2 Large Language Models

2.2.1 Evolution and Capabilities

2.2.2 Prompt Engineering

2.2.3 LLMs in Software Engineering

2.2.4 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 AI-Assisted Software Architecture Design

3.3 DDD-Based Monolith Decomposition

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

The Secuflows is its own client which can model data workflows.

SecuForms - Secure Forms

Secure web Forms ....

4.1.2 Technical Architecture and Security

Secupass is utilizing AES-256-bit encryption for end-to-end data protection.. Data transmission is secured through TLS 1.3 protocols, with optional zero-knowledge encryption available for maximum security. [FTA25]

4.2 Current hurdles

The company's core software has evolved over the years to accommodate numerous requirements. The main Software grew over the years with increasing business needs..

some functions scaled over the years with no supporting architecture behind it making parts of it hard to maintain. The biggest issue is current Architecture.

4.2.1 Current Monolith Architecture

Currently, a substantial portion of the service consists of a large monolithic structure that has become increasingly difficult to maintain. To ensure future readiness and sustain a reliable, robust software foundation, the development team is continuously working to transform this monolith into a more modular architecture (modulith).

To accomplish this transformation, developers—alongside their regular tasks of implementing new features and fixing bugs—decompose individual parts of the software where possible into separate bounded contexts using domain-driven-design (DDD) as presented by Vernon [Ver13, p.62] and explained in Section 2.1. Through this thesis, we aim to investigate how Large Language Models (LLMs) can be effectively utilized to accelerate and improve FTAPI's modularization process. Specifically, we will explore how LLMs can assist software architects in identifying potential bounded contexts, defining domain models within those contexts, and establishing appropriate interfaces between them. This assistance has the potential to significantly advance FTAPI's architectural evolution toward a more maintainable, modular system based on sound domain-driven principles.

5 Methodology

what will be done, how it is planned what steps are taken.

5.1 Research Design

This study employs a mixed-methods approach combining observational analysis, experimental implementation, and comparative evaluation. The research is structured as a three-phase methodology designed to address the core research questions regarding the effectiveness of LLM-assisted domain modeling in complex software architecture transitions.

5.2 Phase 1: Observational Baseline Assessment

This phase establishes a comprehensive baseline understanding of the existing architectural challenges and development processes within the FTAPI platform transition.

Current State Analysis

The initial assessment involves:

- **Architecture Documentation Review**: Analyzing existing system documentation, including architectural decision records, system diagrams, and API specifications
- **Code Base Analysis**: Examining the monolithic codebase to identify:
  - Tightly coupled components

  - Shared data models across different business functions
  - Technical debt hotspots
  - Current module boundaries and their effectiveness

5.2.1 Modularization Strategy Documentation

Document FTAPI's current approach to transitioning from monolith to modulith:

- Criteria used for identifying module boundaries
- Success stories (e.g., SecuRooms modularization)
- Failed or challenging modularization attempts
- Resource constraints and timeline pressures

5.3 Phase 2: LLM Selection and Prompt Engineering

5.3.1 LLM Model Selection

Evaluate and select appropriate LLMs based on:

- **Capability Assessment**: Testing models on domain modeling tasks
- **Context Window Size**: Ensuring sufficient capacity for large requirement sets
- **Output Consistency**: Evaluating reproducibility of results
- **Cost-Benefit Analysis**: Balancing performance with operational costs

Models to be evaluated include:

- GPT-4 (OpenAI)
- Claude 4 (Anthropic)
- Open-source alternatives (e.g., Llama 2, CodeLlama)

5.3.2 Prompt Engineering Framework

Develop a systematic approach to prompt design:

Initial Prompt Templates

Create base prompts for:

- Bounded context identification
- Domain model extraction
- Ubiquitous language detection

Iterative Refinement Process

1. Test prompts with sample requirements
2. Analyze output quality and consistency
3. Refine based on domain expert feedback
4. Document prompt evolution and rationale

Context Injection Strategies

- Determine optimal ways to provide system context
- Balance between comprehensive information and token limits
- Develop chunking strategies for large requirement sets

5.4 Phase 3: Domain Model Generation

5.4.1 Experimental Setup

Test Case Selection

- Identify 2-3 representative modules from FTAPI platform
- Select modules with varying complexity levels
- Ensure coverage of different business domains

Input Preparation

- Compile comprehensive requirement documentation
- Include user stories, acceptance criteria, and business rules
- Prepare existing code snippets and API definitions

5.4.2 LLM-Assisted Generation Process

Bounded Context Discovery

- Feed requirements to LLM with crafted prompts
- Generate multiple candidate bounded contexts
- Document reasoning provided by LLM

Domain Model Creation

For each identified context, generate:

- Entities and value objects

- Aggregates and aggregate roots
- Domain services
- Repository interfaces

Ubiquitous Language Extraction

- Identify key domain terms
- Create glossaries for each bounded context

5.4.3 Manual Baseline Creation

Parallel to LLM generation, experienced DDD practitioners will:

- Analyze the same requirements independently
- Create bounded contexts and domain models
- Document their decision-making process

5.5 Evaluation Approach

tbd.

6 Implementation

Do the parts and document it

6.1 Phase 1 Implementation: Baseline Assessment

6.1.1 Current Architecture Analysis of the BBoM

Understanding the architectural challenges facing FTAPI's modularization efforts required a comprehensive analysis of the existing monolithic codebase, which developers internally refer to as the "Big Ball of Mud" (BBoM). This analysis took place during June 2025 using a combination of static code analysis tools and manual code inspection.

Code Metrics and Structure

SonarQube was used to examine the core monolith consisting of approximately 101,800 lines of Java code. The analysis revealed:

6.2 LLM Setup and Configuration

6.3 Prompt Design and Refinement

6.4 Domain Model Generation Process

6.5 Case Studies

6.5.1 Case Study 1: [First Module/Subdomain]

6.5.2 Case Study 2: [Second Module/Subdomain]

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

List of Figures

List of Tables

Bibliography

[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`.

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

[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.