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

Acknowledgments

Abstract

Contents

Acknowledgments iii

Abstract iv

1 Introduction and Overview 1
- 1.1 Motivation 1
  - 1.1.1 Outlook 2
- 1.2 Research Question and Objectives 2

2 Theoretical Background 3
- 2.1 Domain Driven Design 3
  - 2.1.1 Ubiquitous Language 3
  - 2.1.2 Domain 4
- 2.2 Large Language Models 4
  - 2.2.1 Evolution and Capabilities 4
  - 2.2.2 Prompt Engineering 4
  - 2.2.3 LLMs in Software Engineering 4
  - 2.2.4 Limitations and Challenges 4

3 Related Work 5
- 3.1 Automated Domain Model Generation 5
- 3.2 AI-Assisted Software Architecture Design 5
- 3.3 DDD-Based Monolith Decomposition 5
- 3.4 Research Gap?? 5

4 Study Context: FTAPI Software GmbH 6
- 4.1 Company and Product Overview 6
- 4.2 Current Monolith Architecture 6
- 4.3 Modularization Strategy 7
- 4.4 Development Process and Challenges 7

5 Methodology **8**
5.1 Research Design 8
5.2 Phase 1: Observational Baseline Assessment 8
5.3 Phase 2: LLM Selection and Prompt Engineering 8
5.4 Phase 3: Domain Model Generation 8
5.5 Evaluation Approach 8

6 Implementation **9**
6.1 LLM Setup and Configuration 9
6.2 Prompt Design and Refinement 9
6.3 Domain Model Generation Process 9
6.4 Case Studies 9
6.4.1 Case Study 1: [First Module/Subdomain] 9
6.4.2 Case Study 2: [Second Module/Subdomain] 9

7 Results and Evaluation **10**
7.1 Generated Domain Models and Bounded Contexts 10
7.2 Expert Evaluation Results 10
7.3 Comparison with Manual Domain Modeling 10
7.4 Analysis of LLM Performance 10
7.5 Discussion of Findings 10

8 Discussion **11**
8.1 Impact on Domain Modeling Process 11
8.2 Strengths and Limitations 11
8.3 Implications for Practice 11

List of Figures **12**

List of Tables **13**

Bibliography **14**

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 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.

2.1.2 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."

2.2 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

3.2 AI-Assisted Software Architecture Design

3.3 DDD-Based Monolith Decomposition

3.4 Research Gap??

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

4.2 Current Monolith Architecture

The company's core software has evolved over the years to accommodate numerous requirements. 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.

4.3 Modularization Strategy

4.4 Development Process and Challenges

5 Methodology

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

5.1 Research Design

5.2 Phase 1: Observational Baseline Assessment

5.3 Phase 2: LLM Selection and Prompt Engineering

5.4 Phase 3: Domain Model Generation

5.5 Evaluation Approach

6 Implementation

Do the parts and document it

6.1 LLM Setup and Configuration

6.2 Prompt Design and Refinement

6.3 Domain Model Generation Process

6.4 Case Studies

6.4.1 Case Study 1: [First Module/Subdomain]

6.4.2 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

4.1 FTAPI Software GmbH Logo 6

List of Tables

Bibliography

[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.

[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`.

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