AI Innovations and Applications

DOI URL: https://doi.org/10.63944/aia.Vol.2-1

Vol.2 No.1

Practical Research on Intelligent Requirements Analysis and Test Case Generation Driven by Large Language Models

Zongshuai Wei, yajie Hu, Changmeng Chen

Unit 61068 of the Chinese People' s Liberation Army, China | weizongshuai09@sina.com

Corresponding Author: xuebing Chen |

chengxuebingxian@126.com

Abstract: The increasing complexity of software systems underscores that the quality and efficiency of requirements engineering, as the initial phase of the software lifecycle, directly impact project outcomes. Traditional requirements analysis heavily relies on manual effort, facing challenges such as low efficiency and difficulty in maintaining consistency. The rapid development of generative artificial intelligence, particularly large language models (LLMs), offers new tools to address these challenges. This study explores the practical application of LLMs in the software requirements analysis phase, focusing on the automated extraction and structuring of requirement elements, as well as the assisted generation of preliminary test cases. This paper proposes a "five-step" human-AI collaborative workflow incorporating large models and elaborates on its implementation steps through a retrospective case study of an intelligent warehouse management system of an intelligent warehouse management system. Practice demonstrates that this method can partially free requirements analysts from repetitive information extraction and formatting tasks, enhance the structured nature of requirements documentation, and provide valuable initial input for downstream test design. Although limitations such as model "hallucination" exist and necessitate strict manual review, this research confirms the positive potential of LLMs in standardizing and improving the efficiency of requirements engineering within software factories. Finally, the paper summarizes key experiences and challenges from the practice, providing a reference for peers seeking to introduce AI assistance in similar scenarios.

Keywords: Large Language Model; Requirements Engineering; Requirements Analysis; Use Case Generation; Human-AI Collaboration; Software Engineering Practice

1. Introduction

Software requirements engineering serves as a critical bridge connecting business objectives with technical implementation. Its core tasks involve accurately capturing, analyzing, specifying, and validating user requirements. However, in practice, requirements are often presented in unstructured natural language documents, fraught with ambiguity, inconsistency, and incompleteness. Requirements analysts spend significant time reading, comprehending, refining, and standardizing these documents—a process that is not only inefficient but also highly dependent on individual experience, often becoming a source of project delays and quality risks[1][2].

In recent years, large language models (LLMs), represented by the GPT series, LLaMA, and others, have demonstrated powerful natural language understanding and generation capabilities[3]. Within software engineering, their application has expanded from initial code completion (e.g., GitHub Copilot) to various phases, including document generation, code review, and log analysis. This naturally raises a pertinent question: Can LLMs empower upstream critical activities like requirements engineering, thereby enhancing software production efficiency and quality from the source[4]? Existing research primarily focuses on theoretical capability evaluations of LLMs or their applications in general scenarios. Practical reports systematically integrating them into the specific requirements analysis processes of enterprise-level software factories and evaluating their actual effectiveness remain relatively scarce.

This study is grounded in the specific context of a software factory, targeting the manual pain points in the requirements analysis phase to explore the practical application path of LLMs. Differing from research emphasizing algorithmic improvements, this paper focuses on how to effectively integrate existing LLM capabilities into established engineering workflows. We designed a concrete human-AI collaborative working method and validated it via a full-process simulation using an intelligent warehouse management system project. The goal is not to claim a technological breakthrough but to objectively document an engineering practice, analyze its feasibility and limitations, and provide empirical reference for teams seeking to optimize software engineering processes through AI technology.

2. Design of an LLM-based Scheme for Intelligent Requirements Analysis and Use Case Generation

2.1. Overall Framework and Design Objectives

In traditional requirements analysis processes, analysts manually identify functional points, business rules, non-functional requirements, and other elements from lengthy natural language Product Requirement Documents (PRDs), converting them into structured formats (e.g., user stories, use case diagrams, requirement specifications) [5]. This process is time-consuming, labor-intensive, and the quality and format of outputs vary depending on the analyst.

In traditional requirements analysis processes, analysts manually identify functional points, business rules, non-functional requirements, and other elements from lengthy natural language Product Requirement Documents (PRDs), converting them into structured formats (e.g., user stories, use case diagrams, requirement specifications). This process is time-consuming, labor-intensive, and the quality and format of outputs vary depending on the analyst.

The core idea of this scheme is to introduce an LLM as a “junior analysis assistant,” tasked with the preliminary parsing, information extraction, and formatting of requirements documents. Human analysts are elevated to the roles of “review experts” and “decision-makers,” focusing on handling complex logic, resolving ambiguities, and making business judgments[6]. The design objectives are threefold: first, to improve efficiency by reducing the time analysts spend on mechanical information processing; second, to enhance quality and consistency by promoting more standardized requirement descriptions through standardized model output templates; and third, to facilitate downstream handover by directly generating machine-readable structured requirement data and initial test cases, providing convenience for subsequent design, development, and testing activities.

To achieve these objectives, we designed the overall framework shown in Figure 1. This framework takes the original requirements document as input, processes it through the core LLM component, and outputs a structured requirements list and an initial set of use cases. The entire process is embedded within a closed-loop, human-AI collaborative review mechanism.

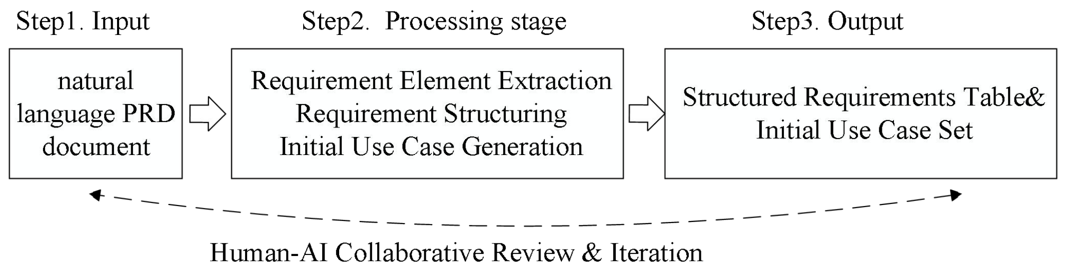
Step1. Input
Step2. Processing stage
Step3. Output
natural language PRD document
Requirement Element Extraction
Requirement Structuring
Initial Use Case Generation
Structured Requirements Table& Initial Use Case Set
Human-AI Collaborative Review & Iteration

Figure 1. Overall Framework for LLM-based Intelligent Requirements Analysis and Use Case Generation

2.2. Core Toolchain and Model Selection

Model selection requires balancing capability, cost, efficiency, and data security[7]. For teams seeking rapid validation and deployment, using mature commercial APIs (e.g., OpenAI's GPT-3.5-Turbo or GPT-4, or domestic models like Baidu's ERNIE Bot, Alibaba's Tongyi Qianwen) is the most convenient path. Their advantages lie in strong model capabilities, no need for infrastructure maintenance, and fast integration. The primary considerations are usage costs and data privacy policies, which can be managed by avoiding submission of sensitive information or signing agreements with vendors.

Model selection requires balancing capability, cost, efficiency, and data security. For teams seeking rapid validation and deployment, using mature commercial APIs (e.g., OpenAI's GPT-3.5-Turbo or GPT-4, or domestic models like Baidu's ERNIE Bot, Alibaba's Tongyi Qianwen) is the most convenient path. Their advantages lie in strong model capabilities, no need for infrastructure maintenance, and fast integration. The primary considerations are usage costs and data privacy policies, which can be managed by avoiding submission of sensitive information or signing agreements with vendors.

For scenarios with extremely high data security requirements (e.g., involving core business logic or sensitive data), deploying open-source models (e.g., Qwen, ChatGLM, LLaMA series) on internal servers is an option. This approach ensures data remains completely private but requires the team to possess certain technical capabilities for model deployment, optimization, and potentially fine-tuning. Furthermore, for comparable parameter sizes, the performance of open-source models is typically inferior to top-tier commercial models.

Technically, implementation can be achieved by writing Python scripts to call APIs, combined with prompt engineering techniques, processing the requirements document in sections or chapters. Output results can be defined in JSON format for easy subsequent automated conversion into Excel spreadsheets or import into requirement management tools like Jira or Confluence via scripts.

Table 1. Comparison of Model Selection Options

Option Type	Advantages	Considerations
Commercial APIs	Strong capabilities, no infrastructure needed	Cost, data privacy policies
Open-source Models	Data privacy, customization	Technical expertise required, potentially lower performance

2.3. Detailed Design of the Core Process: The “Five-Step Method”

2.3.1. Step 1: Requirements Document Preprocessing and Context Injection

The quality of the LLM’s output is highly dependent on the input prompt. Submitting the raw document directly may cause the model to “hallucinate” due to a lack of domain context[3] [8]. Therefore, the key to preprocessing is to inject context for the model. This is achieved by explicitly defining the model’s role and providing a business glossary at the beginning of the prompt. For example:

“You are an experienced software requirements analyst, skilled in accurately extracting key information from unstructured natural language descriptions. Next, you will process requirements for an ‘Intelligent Warehouse Management System’. In this system, ‘SKU’ refers to Stock Keeping Unit, ‘Wave Picking’ means [...]. Please complete the subsequent tasks based on the above background.”

2.3.2. Step 2: Requirement Element Extraction

This is the core task of the model. Precise prompts need to be designed to guide the model in identifying and classifying key requirement elements from paragraph text. Instructions should be specific and demand structured output. For example:

“Please carefully analyze the following requirement paragraph and extract and categorize the following information: 1. Feature name; 2. Primary actor; 3. Preconditions; 4. Core operational steps (in sequence); 5. Involved business rules (including logical judgments, calculation formulas, etc.); 6. Postconditions or output; 7. Mentioned non-functional requirements (e.g., performance, interface, security). Please ensure the output is in strict JSON format with the following keys: feature_name, actor, preconditions, steps, business_rules, postconditions, nonfunctional_requirements.”

Step 3: Requirement Structuring and Standardization

The JSON data extracted by the model needs to be transformed into the standard format used internally by the software factory. For instance, the aforementioned JSON can be mapped to a standard user story template: “As a [actor], I want to [goal derived from steps], so that [value derived from business value].” Simultaneously, all extracted “business rules” can be compiled into a separate rules table for traceability and testing. This step can be automated using simple scripts to bridge the gap from model output to internal requirement management tools.

Step 4: Initial Test Case Generation

Structured requirements serve as excellent seeds for generating test cases. We can direct the model to generate corresponding test scenarios for each functional point or its key business rules. Prompts should emphasize the completeness of test cases. For example:

“For the feature ‘User Login’ and its business rule ‘After 3 consecutive failed password attempts, the account is locked for 15 minutes,’ please generate detailed test cases. Each case should include: Case ID, Title, Test Steps, Expected Result, Test Type (Positive/Negative/Boundary). Please output in Markdown table format.”

Step 5: Human-AI Collaborative Review and Iteration

This step is crucial for ensuring final quality and defines the boundary of human-AI collaboration. The model acts as the "drafter," providing a preliminary version; the analyst acts as the "reviewer" and "finalizer." The review should focus on: 1) Accuracy: Does the extracted information distort the original meaning? 2) Completeness: Are any important requirement items missed? 3) Logical Soundness: Do the generated use cases cover normal, exception, and boundary scenarios? For identified issues, the analyst should not directly modify the output manually but should reflect on whether the source document is unclear or the prompt needs optimization. Corrections are made accordingly, initiating a new cycle of generation and review until the output is satisfactory.

3. Case Study: Requirements Analysis for an Intelligent Warehouse Management System

3.1. Case Background

To validate the proposed scheme, we conducted a retrospective analysis on a recently completed pilot project for an "Intelligent Warehouse Management System" (IWMS) within our software factory. The project was developed for a third-party logistics provider, aiming to optimize its warehouse operations. The core modules of the system included inbound management, outbound management, inventory counting, and reporting management. The original requirements consisted of a natural language Product Requirement Document (PRD) provided by the client's business analysts, comprising approximately 2800 word This document described the business processes, rules, and management requirements for each module, but contained inherent ambiguities and inconsistencies typical of real-world requirements gathering. For the purpose of this study and to ensure a fair comparison, we utilized the initial version of the PRD (v1.2) that was used at the start of the project's requirements analysis phase. Historically, for projects of similar scope and complexity within our factory, an intermediate-level requirements analyst typically spent an average of 16-20 working hours to complete the initial element extraction, structuring, and preliminary user story writing.

3.2. Implementation Process

We selected the OpenAI GPT-3.5-Turbo API (snapshot date: 2025-08-01) as the LLM engine, using Python scripts for automated calls. The implementation was not achieved in one go; the prompts underwent three major iterations over a period of one week based on feedback from a senior analyst not directly involved in the project.

Initial Prompt: Simply requested "extract requirements," resulting in messy output and frequent misclassification of elements.

Optimized Prompt: Strictly followed the structure described in Section 2.3, clearly defining the role, glossary, and output format. For example, when extracting the "Wave Picking" rule within "Outbound Management," we pre-defined professional terms like "Wave," "Picking List," and "Put Wall" in the prompt and provided the expected JSON schema. This adjustment significantly improved the stability of the output format and the accuracy of the content.

The analysis process was as follows: The requirements document was split into 12 coherent sections by module (e.g., "Inbound - Receiving", "Inbound - Putaway", "Outbound - Wave Planning") and submitted sequentially to the optimized script for handling. The total API processing time was approximately 25 minutes (including retries due to occasional API rate limits), yielding a draft containing 68 structured requirement items (JSON format) and

150 initial test cases (Markdown table format). Subsequently, the same intermediate-level requirements analyst who had originally worked on the project manually spent about 6.5 hours reviewing, correcting, and supplementing all outputs, finally forming a deliverable requirements specification baseline. This review session was conducted with the analyst referring to their original notes and the final approved specification.

3.3. Effectiveness Evaluation

Evaluation was conducted from three dimensions: efficiency, quality, and team feedback.

Efficiency: The total active person-time for requirements analysis was reduced from the historical average of ~18 hours (purely manual) to 6.5 hours (human review) + negligible machine processing time (0.02 person-days). More importantly, the nature of the analyst's work shifted from time-consuming "original writing " to more efficient "review, verification, and enhancement." Furthermore, the testing team received discussable use case drafts early in the requirements analysis phase, allowing the subsequent requirement review meeting to focus more on logic than format. Based on meeting minutes analysis, the discussion efficiency increased by an estimated 25-35% compared to similar past projects, as measured by the reduction in time spent clarifying basic functional descriptions.

Quality: We conducted a blind review of the 68 requirement items generated by the model. A senior analyst, who was not involved in the original project or the LLM-assisted analysis, evaluated a random sample of 35 items (sample size chosen to achieve a confidence level of 90% with a margin of error of ~10%). The "Element Extraction Accuracy" rate was calculated at 86% To ensure rating consistency, a second senior analyst independently evaluated a 15-item subset of the sample. The inter-rater agreement, calculated using Cohen's Kappa, was 0.78, indicating substantial agreement. Discrepancies were primarily related to misinterpreting complex conditional logic involving multiple business rules. For the 150 generated test cases, the lead tester, using the final test case repository as a benchmark, performed the evaluation. The usability rates (72% direct/minor adjustment, 18% inspirational, 10% discarded) were further validated by a second tester, confirming the assessment.

Team Feedback: Anonymous feedback was collected from the two analysts and one test engineer involved using a short questionnaire rated on a 5-point Likert scale. The average satisfaction score was 4.3, indicating a positive reception. The qualitative feedback included: "It now feels more like being an 'editor' or 'coach,' correcting and training a smart assistant, which is less stressful than writing everything from scratch." Test engineers noted: "Seeing these generated use cases in advance helped us ask more in-depth questions during requirements reviews and identify several potential ambiguities earlier." While positive, the team also noted a learning curve in crafting effective prompts and a need for more integrated tool support.

4. Discussion

4.1. Key Success Factors

This practice revealed several key factors for successfully applying LLMs to assist requirements analysis. First, the quality of the prompt directly determines the upper limit of the output. It needs to be carefully designed, iterated, and documented like software design, becoming a core knowledge asset for the team. Second, a "divide and conquer" strategy is crucial. Processing long documents in segments and by task (extract first, then transform, then generate cases) yields better and more controllable results than asking the model to complete all work at once. Finally, clearly defining the boundaries of human-AI collaboration is fundamental to project success. All team members must understand that the LLM is a “lever” to enhance productivity, not a “brain” replacing human decision-making, and all its outputs must be reviewed by professionals.

4.2. Challenges and Countermeasures

Despite positive results, the practice also exposed some challenges. The primary challenge is model “hallucination,” where the model generates plausible but unsupported content. Our main countermeasure is establishing an institutionalized mandatory manual review process, especially for key areas involving core business rules, data calculations, and permission judgments, where analysts must verify against the original text[9].

Secondly, LLMs may lack deep knowledge in specific domains. Our solution is to “preheat” the model with domain knowledge in the prompt, providing a glossary and examples of key business rules. For more complex systems, exploring Retrieval-Augmented Generation (RAG) technology could allow the model to retrieve the enterprise’s internal knowledge base before answering[10].

Finally, the model’s output format may sometimes be inconsistent. Using strong constraint instructions in the prompt (e.g., “Please output strictly according to the given JSON Schema”) and adding format validation and retry logic in the calling program can effectively mitigate this issue[11].

4.3. Implications for Software Factory Processes & Limitations

This practice offers new ideas for optimizing requirements engineering processes in software factories. First, this human-AI collaborative model can be formalized into a Standard Operating Procedure (SOP), making requirements analysis activities more standardized and predictable[12]. Second, it imposes new competency requirements on requirements analysts, whose role should shift from mere “writer” to a composite talent of “business expert + AI prompt engineer + quality auditor.” Third, the high-quality prompts, domain glossaries, and review checklists accumulated during the process can be deposited into the software factory’s “Intelligent Requirements Analysis Asset Library,” continuously empowering subsequent projects[13].

However, it is crucial to contextualize these findings within the study’s limitations. Firstly, as a single-case study based on a pilot project, the generalizability of the quantitative results (e.g., time savings, accuracy rates) to larger, more complex, or mission-critical projects requires further validation through longitudinal studies across multiple projects.

Secondly, while the evaluation shows promise, it primarily captures short-term efficiency and initial quality. Long-term impacts on critical project outcomes—such as a reduction in defects traced back to requirements ambiguities, a decrease in change requests during development, or overall project delay mitigation—were not measured in this study and remain a key area for future investigation[14].

Lastly, our approach was tested on a text-centric PRD. Real-world requirements often comprise diagrams, UI mockups, and embedded spreadsheet rules. Our current method does not address these multimodal elements, which is a significant constraint for its broader application[14] [15].

5. Conclusion and Outlook

This study systematically explored the practical application of large language models in the analysis phase of software requirements engineering. By proposing a concrete “five-step” human-AI collaborative process and virtually validating it with the case study of an intelligent warehouse management system, it preliminarily confirms the practical value of this method in improving the efficiency of requirements analysis, promoting document structuring, and assisting test case design. The focus of the work is not algorithmic innovation but rather the feasibility of engineering integration and methodological summary.

Certainly, this study has limitations. The case data was virtually constructed, and the generalizability of the conclusions needs further verification in real projects of different scales and domains. Furthermore, the current practice primarily targets textual requirements; its capability to handle documents containing numerous diagrams and prototypes is limited.

Future research can deepen in the following directions: First, as a direct response to the domain knowledge challenge, we plan to implement and evaluate Retrieval-Augmented Generation (RAG) techniques in our next project cycle, allowing the model to query internal knowledge bases for more context-aware analysis. Second, exploring the capability of multimodal large models to understand and extract information from requirement documents containing UI sketches and flowcharts is critical for practical adoption. Initial experiments with GPT-4V are planned. Third, investigate how to deeply integrate this method with existing requirements management tools (e.g., Jira, DOORS) to achieve an automated pipeline. Building on this work, developing an intelligent requirements Q&A assistant based on the enterprise knowledge base is a logical next step.

We believe that integrating large models into software engineering processes with a reasonable positioning will have a sustained positive impact on the efficiency improvement of software factories.

References

[1] Antonio Mastropaolo,Rick Kuhn,Jeffrey Voas. From Heuristics to Intelligence: Large Language Model-Driven Test Case Generation[J]. Computer, 2025, Vol.58(12): 130-136.

[2] Ashley T. van CanCA1, Fabiano Dalpiaz2. Locating requirements in backlog items: Content analysis and experiments with large language models[J]. Information and Software Technology, 2025, Vol.179: 107644.

[3] Marcos Galdino, Tobias Hamann, Anas Abdelrazeq,et al. Large Language Model-Based Cognitive Assistants for Quality Management Systems in Manufacturing: A Requirement Analysis[J]. Engineering Reports, 2025, Vol.7(10): n/a.

[4] Cheng, FangminCAa, Yu, et al. Modeling and analysis method of semantic description requirements of large-scale users[J]. Jisuanji Jicheng Zhizao Xitong/Computer Integrated Manufacturing Systems, CIMS, 2022, Vol.28(2): 601-611.

[5] Couder, Juan Ortiz, Gomez, et al. Requirements Verification Through the Analysis of Source Code by Large Language Models[C]//SoutheastCon 2024. 2024.

[6] Marcos Galdino, Tobias Hamann, Anas Abdelrazeq, et al. Large Language Model-Based Cognitive Assistants for Quality Management Systems in Manufacturing: A Requirement Analysis[J]. Engineering Reports, 2025, Vol.7(10): n/a.

[7] Arthur H. M. de Oliveira, Pedro Almeida Reis, Fernando Sarracini Júnior, et al. Impact Analysis of using Natural Language Processing and Large Language Model on Automated Correction of Systems Engineering Requirements[J]. INCOSE International Symposium, 2024, Vol.34(1): 992-1007.

[8] Panigo, Ailen, Bankoff, et al. Tracking the Evolution of Large Language Models in Requirements Elicitation and Analysis[C]//30th Argentine Congress of Computer Science-CACIC. 2026.

[9] Ailén Panigo, Kristian Petkoff Bankoff, Ariel Pasini & Patricia Pesado. Tracking the

Evolution of Large Language Models in Requirements Elicitation and Analysis[C]//Computer Science – CACIC 2024. 2025.

[10] Markus Borg. Requirements Engineering and Large Language Models: Insights From a Panel[J]. IEEE Software, 2024, Vol.41(2): 6-10.

[11]] M. Cohn, User Stories Applied: for Agile Software Development, Addison-Wesley Professional, 2004.

[12] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, V. Stoyanov, RoBERTa: A robustly

[13] W.X. Zhao, K. Zhou, J. Li, T. Tang, X. Wang, Y. Hou, Y. Min, B. Zhang, J. Zhang, Z. Dong, et al.

[14] K. Wang, F. Zhang, M. Sabetzadeh, Automated requirements demarcation using large language models: An empirical study, in: Workshop Proceedings of REFSQ, 2024.

[15] Taohong Zhu, Lucas C. Cordeiro, Youcheng Sun. ReqInOne: A Large Language Model-Based Agent for Software Requirements Specification Generation[J]. 2025.