Proceedings of the 20th Conference on Computer Science and Intelligence Systems (FedCSIS) pp. 121–132 DOI: 10.15439/2025F8736 ISSN 2300-5963 ACSIS, Vol. 43

A Framework for Model-Driven AI-Assisted Generation of IT Project Management Plan and Scope Documents

Jānis Rihards Blaževičs
Riga Technical University,
Institute of Information Technologies,
Riga, Latvia
Email: janis-rihards.blazevics@edu.rtu.lv

Oksana Nikiforova
0000-0001-7983-3088
Riga Technical University,
Institute of Information Technologies,
Riga, Latvia
Email: oksana.nikiforova@rtu.lv

Oscar Pastor
0000-0002-1320-8471
Universitat Politècnica de València, Valencia, Spain
Email: opastor@dsic.upv.es

Abstract—**This paper presents a model-driven AI-assisted approach for generating IT project management plan and scope documents, aiming to improve efficiency and quality in software development projects. Effective documentation in the early project phase is critical, yet often resource-intensive. The proposed solution consolidates best practices from widely used project management methodologies and standards to create a dynamic, adaptable framework for document generation. The study identifies key components and input data required for generating high-quality plans using model transformations and generative AI. A prototype supporting the solution is developed featuring a local processing engine, integrated with a large language model, a vector database, and an embedded model.**

Index Terms—**IT project management, project documentation, model-driven development, generative AI, prompt engineering, software development automation, large language models, AI-assisted tools.**

I. Introduction

IN MODERN IT project development, the demand for accelerated delivery cycles continues to increase, driven by competitive pressure, agile practices, and customer expectations [1]. One of the most critical processes in the early phases of the project lifecycle is the development of project documentation [2]. Well-structured documentation, such as project management plans and scope statements, forms the basis for mutual understanding between clients and delivery teams, and contributes directly to project success and client satisfaction [3].

To address the need for speed without compromising quality, automation tools have been introduced to support documentation processes. These tools can generate content based on reusable templates, domain knowledge, and recognized best practices [4]. However, most existing solutions act as passive assistants, requiring significant manual effort to ensure completeness, consistency, and compliance with professional standards [5].

Since the emergence of generative artificial intelligence (AI) technologies around 2020 [6], there has been growing interest in using AI to automate the creation of project documentation. Generative models such as large language models (LLMs) [7] offer promising capabilities for producing structured, domain-specific texts [8]. However, researchers should overcome challenges related to the need for precise prompt formulation, ensuring contextual relevance, managing data quality, and aligning with industry standards [9]. Addressing these challenges requires not only technical solutions but also methodological guidance.

This raises a fundamental research question: Is it possible to integrate generative AI into documentation development without compromising quality and compliance? This paper investigates this question by proposing an AI-assisted solution grounded in software engineering principles, aiming to maintain documentation integrity while leveraging the potential of generative AI. The research goal of the results presented in this paper is to develop an AI-assisted solution for generating IT project management plan and scope documents applicable in software development projects, by consolidating best practices in project documentation as defined across various development methodologies and industry standards.

The paper is organized as follows. The next section discusses the research background and summarizes the related literature. Section 3 presents the foundations used in the solution and Section 4 explains the essence of the solution. Section 5 demonstrates the practical application of the solution and comments its validation and improvements. The final section summarizes the main research findings and outlines directions for future work.

This research has been supported by Research and Development grant No RTU-PA-2024/1-0015 under the EU Recovery and Resilience Facility funded project No. 5.2.1.1.i.0/2/24/I/CFLA/003 "Implementation of consolidation and management changes at Riga Technical University, Liepaja University, Rezekne Academy of Technology, Latvian Maritime Academy and Liepaja Maritime College for the progress towards excellence in higher education, science, and innovation".

II. Background and Related work

Since the emergence of generative artificial intelligence (AI) technologies around 2020, such as OpenAI's GPT-3 [6], there has been growing optimism that AI could dramatically streamline project documentation by enabling the automatic generation of high-quality documents in just a few clicks. The rapid advancement of LLMs has fueled expectations that these tools could reduce the manual effort required for creating project specifications, requirement documents, and other critical deliverables. However, several significant challenges remain in realizing this vision fully:

1. Generative AI often struggles to capture the specific organizational context, domain terminology, and stakeholder expectations that are essential for meaningful documentation [9].
2. Effective AI-generated documentation depends heavily on the availability of structured, high-quality input data. In many real-world projects, this data may be incomplete, ambiguous, or inconsistent [10].
3. Automatically generated content can raise concerns about traceability, authorship, and the responsibility for errors or omissions in critical documents [11].
4. Industry-specific methodologies and compliance requirements (e.g., ISO, IEEE) can be difficult to encode into generic AI systems without significant customization [12].
5. Even advanced generative tools require skilled human supervision to ensure accuracy, relevance, and alignment with project goals [13].

These limitations suggest that while generative AI has strong potential to support project documentation, its role is currently best viewed as augmentative rather than fully autonomous. Future developments may narrow these gaps, but effective human-AI collaboration remains essential for reliable outcomes in IT project documentation.

Recent studies have demonstrated the growing potential of generative AI in enhancing project management processes, particularly in the areas of documentation, planning, and efficiency. One of the key applications of generative AI is the automation of project documentation, including project plans and scope definitions. AI models can consolidate and structure content from various sources to generate standardized and coherent documents, ensuring compliance with project management standards and improving overall consistency [14]. Additionally, generative AI is increasingly used to generate meeting summaries, translate technical content, and update project texts, contributing to more accurate and timely documentation [15].

In project planning, AI tools can support a wide range of functions such as scope definition, scheduling, cost estimation, resource allocation, stakeholder analysis, and risk assessment. Some studies suggest that generative AI can match or even outperform human planners in certain structured planning tasks, while still requiring expert oversight for contextual decisions [16]. Moreover, the integration of AI in Agile methodologies has shown promise in optimizing team workflows, enhancing collaboration, and automating repetitive tasks [17]-[18].

The collaboration between human project managers and AI systems has been identified as a critical factor for success. Key performance indicators for such collaboration include efficiency gains in time, cost, and resource usage, as well as improved quality, clarity, and accuracy of project outputs [19]. Research also highlights the benefits of using generative AI in the early stages of innovation, such as digital prototyping and ideation, where rapid iteration and content generation are crucial [20].

However, several challenges remain. Effective integration of AI into project management environments must address issues such as input data quality, compatibility with legacy systems, and ethical concerns regarding algorithmic transparency and accountability [21]-[22]. Furthermore, while generative AI can produce initial drafts of project documentation, human expertise is still essential to validate and refine the output to ensure relevance, accuracy, and compliance with specific organizational contexts [16].

These findings underscore the dual role of generative AI as both an accelerator and a collaborator in project management. The literature suggests that while generative AI tools provide substantial benefits, their successful implementation relies on structured prompts, methodological support, and continuous human oversight.

To address the limitations of generative AI in project documentation, Nikiforova, *et al.* [23] proposed a solution based on the integration of model-driven development (MDD) principles. By embedding structured models into the AI-assisted documentation process, the approach aims to constrain and guide AI output according to predefined rules, templates, and domain-specific semantics. This model-driven approach helps overcome common issues such as contextual inconsistency, structural errors, and lack of standard compliance. It also enhances traceability and maintainability by linking generated content to formal models that represent project requirements and design logic.

In response to the limitations of generative AI tools, researchers and practitioners have increasingly focused on the field of prompt engineering, which is the practice of crafting and structuring input prompts in a way that guides AI systems to produce higher-quality, more relevant outputs [24]. Rather than relying on generic instructions, prompt engineering emphasizes precise formulations, contextual cues, and example-based guidance to improve the performance of language models in complex tasks such as project documentation. The emerging field of prompt engineering aims to optimize prompt formulation in order to elicit high-quality, contextually accurate responses from generative models [25].

This emerging discipline is particularly relevant in domains like IT project management, where document quality is influenced by terminology accuracy, logical structure, and compliance with standards. By experimenting with different prompt patterns, researchers aim to understand how to systematically elicit responses that align with user expectations and domain-specific requirements.

Early studies and practical experiments indicate that carefully designed prompts can significantly enhance the clarity, completeness, and consistency of AI-generated documents. As a result, prompt engineering is becoming a critical skill for those seeking to integrate generative AI into documentation workflows, bridging the gap between AI capabilities and real-world needs.

The research presented in this paper builds upon the idea of constraining generative AI to reduce hallucinations and uncontrolled variability in generated content. The proposed approach is oriented toward leveraging the strengths of model-driven engineering by generating as much documentation content as possible through formal model transformations [26]. Where model transformation is insufficient, such as in the generation of narrative or descriptive elements, prompt engineering is applied, guided by artifacts from the target model [27]. This ensures that even AI-generated free-text sections remain contextually relevant, structurally aligned, and traceable to defined project components. The overall goal is to balance AI creativity with formal control, reducing risks while maintaining efficiency and adaptability.

III. Research Methodology and Conceptual Framework

This research follows the Design Science Research (DSR) methodology as proposed by Hevner et al. [28], which is widely used for the development and evaluation of IT artifacts. The goal is to create a novel, AI-assisted solution for generating IT project management plan and scope documentation. This artifact integrates model-driven engineering principles with generative AI capabilities to support early-phase documentation in software development projects.

In accordance with Hevner's seven guidelines for DSR, this study:

1. Designs an innovative artifact – a model-driven framework and software prototype for document generation;
2. Addresses a relevant business problem – namely, the inefficiency and inconsistency in early-phase project documentation;
3. Is evaluated through practical application and iterative refinement;
4. Contributes both a utility-focused solution and theoretical insights into the integration of generative AI with structured project management models;
5. Applies rigorous methods for modeling, implementation, and validation;
6. Communicates the research effectively to both academic and practitioner audiences;
7. Clearly situates the research within the context of information systems and project management disciplines.

The proposed approach balances the creative potential of generative AI with the structural rigor of model-driven development. The main objective is to constrain AI-generated content within well-defined semantic and syntactic boundaries, ensuring compliance with established project management standards described in the next subsections.

A. Model-Driven Development Principles

Model-Driven Development (MDD) is a software engineering approach that emphasizes the use of formal models as the primary artifacts throughout the development lifecycle [29]. Instead of writing low-level implementation code directly, developers create abstract models that define system structure, behavior, and logic [30]. These models are then transformed into executable code or other technical artifacts through automated tools. The main goal of MDD is to increase productivity, improve consistency, and reduce human error by shifting development effort to higher levels of abstraction.

MDD was originally developed to address complexity in large-scale systems and to enable better alignment between business requirements and technical implementation. It has proven particularly effective in domains requiring precision, standardization, and traceability, such as embedded systems, enterprise architectures, and critical software solutions.

Based on systematic literature mapping of solutions offered under the MDD idea, Nikiforova *et al.* [31] conducted an in-depth analysis of transformations between key artifacts used in the early phases of IT projects, that is, before the implementation stage. The study focused on identifying how different project artifacts, such as goal models, process models, use case models, domain models, and architectural overviews, can be interrelated through formal transformation rules.

The research highlights that certain artifacts can be automatically derived from others using well-defined transformation methods, significantly reducing manual effort and increasing consistency across documentation. For instance, structured requirements can be transformed into initial system models or project plans with minimal human intervention. However, not all relationships are fully automatable (shown as dotted lines in Fig. 1), some transformations still require manual refinement or validation, especially in cases involving complex contextual interpretation or stakeholder input. Exactly, these transformations can be assisted with the usage of generative AI.

An overview of all these artifact transformations and their interdependencies is illustrated in Fig. 1 [27]. This schema serves as a foundation for the proposed AI-assisted generation process, where transformation rules help define the scope of content that can be automatically constructed and guide the prompt generation strategy for generative AI components.

Although MDD adoption has declined in recent years, partly due to the rise of more lightweight, iterative approaches like Agile [32], the core idea of structuring development around formalized models remains powerful. In the context of generative AI, MDD principles offer a unique advantage: they can provide a well-defined, controlled input structure for guiding AI outputs.

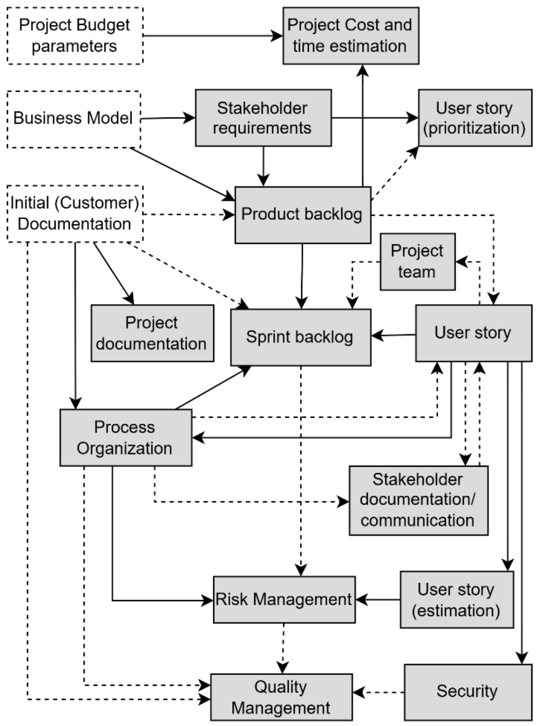

Fig 1. Transformation among IT project artifacts

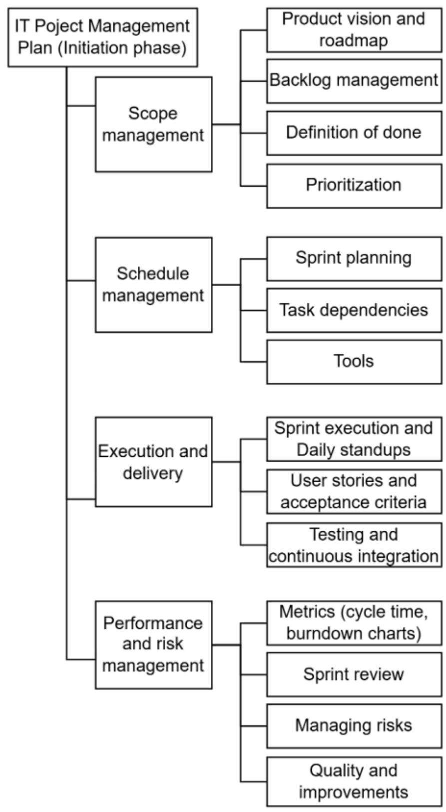

Fig 2. IT project documentation structure and content

This makes it possible to constrain generative models within meaningful boundaries, improving the reliability and quality of generated content. By reintroducing MDD concepts into AI-assisted documentation generation, this research aims to combine the rigor of model-driven processes with the flexibility of modern language models, achieving better control, relevance, and integration in project management documentation workflows.

B. Software Engineering Standards

The implementation of software engineering standards plays a vital role in ensuring consistency, quality, and traceability in IT project documentation. Given the increasing complexity of software development projects and the integration of AI-assisted tools, aligning documentation processes with internationally recognized standards becomes not only beneficial but necessary. The proposed solution leverages the following software engineering standards:

1. ISO/IEC/IEEE 29148:2018 Systems and software engineering — Life cycle processes — Requirement's engineering [33];
2. ISO/IEC/IEEE 16326:2019 Systems and software engineering — Life cycle processes — Project management, 2019 [34];
3. ISO 21502:2020, Project, programme and portfolio management — Guidance on project management, 2020 [35];
4. Corresponding elements of SWEBOK [36];
5. Corresponding elements of PMBOK [37].

Software engineering standards are used in two ways. The first, they define the macrostructure of the document (like in the way presented in Fig. 2) [31].

The standards' instructions are used to guide the structure and content of generated project documents. These standards provide clear expectations regarding the format, terminology, and required content of artifacts such as project charters, requirement specifications, and management plans.

And, the second, the standards shape the micro-level instructions that guide the AI in generating specific paragraphs or sections. Rather than embedding static prompts into the generative AI component, this solution dynamically generates prompts based on structured input, specifically utilizing content derived from model transformations. The AI model is thus directed by prompts that are themselves informed by software engineering standards and documentation guidelines. This ensures that generated outputs are not only linguistically fluent but also semantically aligned with professional project management practices.

For example, prompts for requirement specification generation integrate instructional content drawn directly from ISO/IEC/IEEE 29148:2018, such as the need for completeness, verifiability, traceability, and feasibility.

Additionally, content expected from model transformations, such as stakeholder roles, system goals, and component descriptions, is automatically fed into the prompt generation process. This staged approach enables the AI to act within clearly defined semantic boundaries while maintaining flexibility for natural language output. This mechanism represents a hybrid strategy that combines the strength of model-driven development with the adaptive capabilities of generative AI. By grounding prompt generation in formal standards and structured content, the solution significantly reduces the risk of hallucinations and irrelevant output, ensuring higher reliability and usability of project documentation.

In summary, the integration of software engineering standards serves as a scaffolding for both the structure and semantics of AI-generated project artifacts. It bridges the gap between free-form generative models and formal documentation requirements, resulting in outputs that are both technically valid and aligned with industry expectations.

IV. Solution Design and Approach

This section presents the formal framework and implementation of the proposed approach for AI-assisted generation of project documentation. The solution is built on a hybrid methodology, combining model-driven development (MDD), prompt engineering, and alignment with software engineering standards listed in previous sections.

The proposed solution is based on the principle that structured artifacts (i.e., models, templates, and metadata) should guide document generation. When natural language output is required, prompt engineering is applied, combining structured artifacts and guidance to direct the language model. This hybrid approach reduces hallucinations and improves consistency. The solution operates through several stages:

- **Input Stage:** Project-specific structured artifacts, such as project artifacts, are provided as input. Wherever possible, content is derived from model transformations, using approaches and tools like [38] [39].
- **Instruction Stage:** Documentation structure and content expectations are sourced from industry standards. These sources serve as a foundation for structuring both the overall layout and the content of the generated documentation. Excerpts from standards, such as definitions, guidelines, and section requirements, are used as instructional cues within prompts to ensure that the generated text aligns with professional expectations.
- **Transformation Stage:** Input artifacts and instruction templates are combined through structured transformation logic (defined in JSON), which dynamically generates task-specific prompts.
- **Generation Stage:** These dynamic prompts are used to control a language model (LLM) that generates natural language output. Furthermore, in the proposed solution, the prompts used by the generative AI component are not static or hardcoded. Instead, they are dynamically generated based on software engineering standards and best-practice "instructions," as well as content defined through model transformations. This means that the system first interprets structured input, such as elements derived from project models, and then constructs a tailored prompt to guide the language model.
- **Validation Stage:** Output is validated against structural schemas and evaluated on semantic alignment, completeness, and relevance.

A key innovation is the dynamic generation of prompts based on model-driven and SE standards-based inputs. Instead of using static instructions, the approach uses structured data to generate a prompt, which in turn is used to generate the final documentation text. This "generative AI powering generative AI" approach ensures that outputs are both contextually relevant and semantically accurate, thus allowing precise alignment between structured models and natural language output and reducing the likelihood of irrelevant or hallucinated content. It is like a controlled pipeline that leverages automation while maintaining contextual fidelity and compliance with documentation standards.

The system is implemented in Python using the Ollama framework. A locally hosted LLaMA 3.2:3b model handles language generation, while ChromaDB stores embedded vector representations of source content. The conceptual schema of the solutions architecture is shown in Fig. 3. Following the outlined stages, a generative AI-based solution was developed, structured around three key modules: the Input Module, the Processing Module, and the Output Module.

The **Input Module** validates and processes user-provided files, including contextual documents and control JSONs. Uploaded PDF content is split into fragments using token limits and overlap parameters to preserve context. Each fragment is embedded as a vector and stored in ChromaDB, along with metadata such as file type and source ID. Instruction files are parsed into stepwise JSON objects for controlled processing.

The **Processing Module** executes semantic queries based on the instruction steps. Each step retrieves relevant fragments using vector similarity and uses them as input for the next step. This enables incremental generation of documentation components, where each output builds on previous results. The final output is compiled into structured sections based on a predefined JSON schema.

The **Output Module** formats the completed documentation into a standards-compliant structure, ready for export or integration.

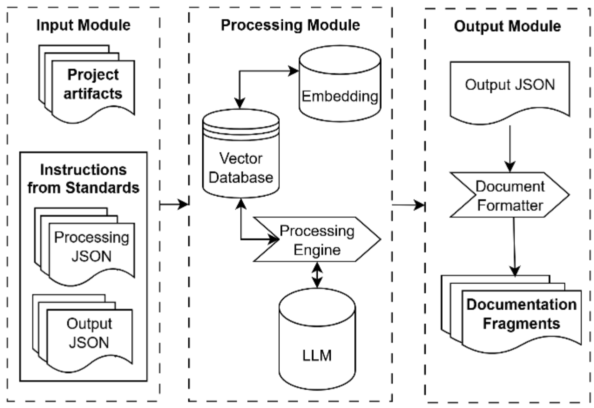

Fig 3. IT project documentation structure and content

The prototype screenshot is shown in Fig. 4. The user interface enables uploading project artifacts and control JSON files based on engineering standards. It supports "Process" and "Generate" actions to trigger analysis and document creation via a lightweight, Python-based framework.

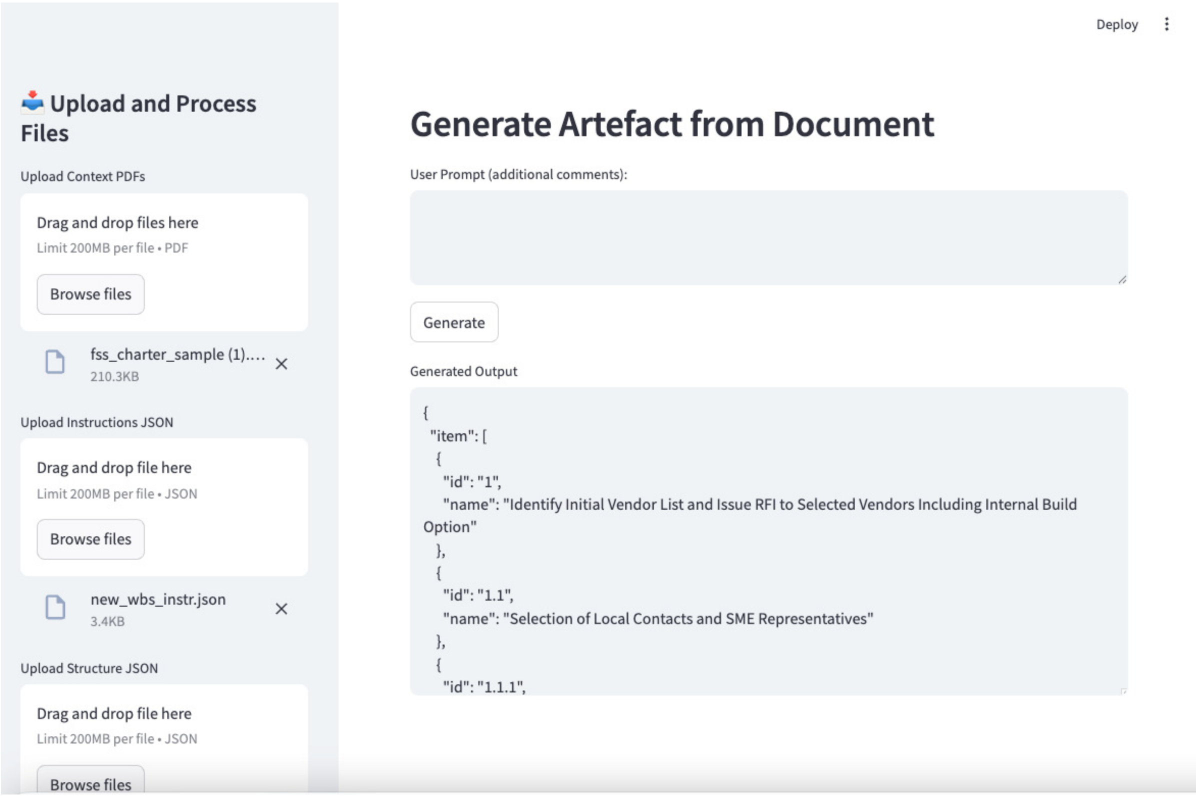

Fig 4. Solutions supporting prototype UI

V. Solution demonstration and Validation

The developed solution focuses on the generation of documentation for the project initiation phase, which primarily are the project scope, requirements, and stakeholder register. However, the overall architecture and methodology are inherently adaptable to other types of software project artifacts (e.g., test strategies, change logs), provided that the step-based instruction approach is followed.

The design of the approach was informed by a critical analysis of limitations in existing AI-based documentation methods [41]. A key technical constraint was the limited token processing capacity of large language models (LLMs). Supplying the model with excessive or poorly structured context, even within token limits, can degrade output quality. This challenge was especially relevant due to the decision to implement the entire solution locally, without access to commercial API-based models.

To overcome these constraints, the solution relies on a hybrid generation process that combines model-driven input with structured prompt engineering techniques. The system employs dynamic prompt construction, where instructions are derived from formal templates and merged with input artifacts. This enables semantic precision and output consistency, while minimizing hallucinations.

A. Prompt Construction and Input Engineering

Prompt engineering is a critical set of techniques aimed at improving the precision, relevance, and structural integrity of instructions provided to large language models (LLMs). Given the inherent token limits and sensitivity to vague or ambiguous inputs, effective prompt design plays a key role in achieving high-quality, context-aware results.

While most modern LLMs can process both natural language and semi-structured formats, generating well-structured outputs, such as formal documentation artifacts, remains challenging. This is especially true when using general-purpose models that are not fine-tuned for technical or procedural content. To address this, researchers and practitioners have explored strategies such as zero-shot prompting, few-shot prompting, chain-of-thought prompting, and input injection, where contextual cues are embedded into the prompt to guide the generation process more effectively.

An increasingly popular and effective technique, particularly in software engineering and academic contexts, is schema-based prompting. In this method, the input is organized into clearly defined data segments or templates, enabling the model to recognize expected content categories and follow predefined output structures. This improves both the model's contextual understanding and its ability to maintain the desired format over longer outputs [41].

In the proposed solution, prompt and input engineering form the core of the generation workflow. Instruction files are parsed into stepwise key-value pairs and stored in session memory. During generation, the system performs iterative prompting, combining each instruction step with previously generated content.

Fig. 5 presents an example of the JSON-based prompt structure, and Figure 6 shows the resulting generated output.

```
<Context>
    1. I am creating a local generative AI solution using Llama 3.2:3b model together with embedding
       model and local vector database ChromDB. The envrionment will not have access to the
       internet.
    2. Model purpose is to be a local generative text assistant that handles input data based on
       provided instructions.
</Context>
<Instructions>
    1. The model will receive following inputs interatively as prompt:
           a. fragments from the main document,
           b. instruction describing what to do with the fragments. This instruction changes for
              each iteration.
           c. Result from previous iterations
    2. The model should only use information received as context fragments or previous results as
       source. Instructions are the task what model should achieve.
    3. Take into account that the prompt cannot be too long due to token limitations of the model if
       used for every interaction.
    4. Write me a coprehensive prompt to use as system prompt for Llama 3.2:3b model to set it's
       main purpose.
</Instructions>
<Examples>
    1. You are a document generator llama 3.2:3b that generates answers based on context.
    2. Main rule : You only can use context and previous outputs as source of information
</Examples>
```

Fig 5. A prompt for prompt generation on GPT-4-mini model

```
### Sistēmas ievade
system_prompt = """
You are a document generation assistant local model llama3.2:3b.
You will receive:
    - <Context>: You will receive text fragments for context input
    - <Instructions>: JSON payload with processing rules under
      'llm_prompt_instructions'.
    - <Output structure>: JSON payload defining 'output_structure'.
    - <User input>: optional additional information.
Tasks:
 1. Process <Context> according to <Instructions> JSON.
 2. Generate output strictly following <Output structure> JSON keys and hierarchy.
 3. If <Context> lacks any required field, insert a placeholder in the form
<MISSING_field_name>.
 4. Do not add information not present in <Context>:.

Master rule: If you do not find information based on specific <Instructions>:. Leave output
empty.
"""
```

Fig 6. The generation result: a prompt for Llama 3.2:3b model

Additionally, semantic retrieval is supported through a local vector database queried via Retrieval-Augmented Generation (RAG). Relevant context fragments are injected into each prompt, ensuring that the model's output is both semantically grounded and contextually accurate.

So far, the instruction is processed as part of the iterative prompting loop, where the system performs the following steps:

1. Retrieves relevant fragments from the vector database using tags such as "stakeholder_goals" and "deliverables".
2. Constructs a prompt that integrates retrieved content, the instruction goal, and global generation rules (e.g., tone, structure).
3. Executes the prompt using the LLM, producing a scoped and semantically grounded section.

The resulting output typically begins with a structured summary of project boundaries, highlights included and excluded deliverables, and clarifies key stakeholder expectations, which all are grounded in the source context provided by the uploaded artifacts.

This process ensures that generated text is not only grammatically correct and coherent, but also aligned with input data, methodologically consistent, and easy to validate. If any relevant data is missing (e.g., no deliverables listed), the model leaves the section incomplete or explicitly notes the absence, avoiding hallucinated content.

The core task defined for the model was text generation based on structured input instructions. An example of the instructions given for the Project Scope is shown in Fig. 7. Crucially, each iteration updates the prompt dynamically, preserving prior results and refining the context. This strategy has shown to be highly effective in reducing hallucinations and ensuring continuity across output sections. The base system prompt (or system message) is used to define global context and behavior for all interactions with the LLM.

To optimize prompt formulation, OpenAI's GPT-4-turbo model (May 2024 version) was used to assist in designing the system-level prompts for the local LLaMA 3.2:3b model. These instructions define the system's role, generation boundaries, and output expectations based on the runtime environment and technologies used in this project [41]-[42].

These instructions clarified the expected format and explicitly emphasized that the context PDF file is the sole source of truth for content generation. The model was also guided to respect the limitations of the local environment and its own processing capacity, discouraging overly long or ambiguous prompts.


```
{
  "instruction_id": "project_scope_statement",
  "document_type": "Project Scope Statement",
  "description": "Using the project charter as input, build a Project Scope
Statement.",
  "input_type": "project_charter",
  "llm_prompt_instructions": {
   "goal": "Create a valid Project Scope Statement JSON, containing only the
elements defined below, drawn solely from the input charter.",
    "steps": [
     "1. Project Overview: Extract from the charter a concise statement of project
purpose, objectives, and expected outcomes. Insert data into ProjectName, high-
level Objectives, SummaryDescription fields.",
     "2. Product Scope Description: Detail what will be built or delivered. List the key
features, functions, and performance characteristics of the final product or service
as described in the charter.Insert data into ProductScopeDescription fields ",
     "3. Deliverable Breakdown: Enumerate each major deliverable the project will
produce. For each deliverable, assign a unique identifier and a brief description. Do
not include any tasks, only end-products or results. Insert data into id, description
fields ",
     "4. Scope Boundaries: Define what is in scope (specific inclusions) and what is
out of scope (specific exclusions). Use two separate lists: one for inclusions and
one for exclusions. Insert data into ScopeInclusions, ScopeExlusions fields",
     "5. Acceptance Criteria: For each deliverable, specify the measurable conditions
or tests that must be satisfied for formal acceptance. Insert data into deliverableId,
criteria fields if data exists. If missing, use '<MISSING_AcceptaneCriteria> '.",
     "6. Assumptions: List all assumptions from the charter (conditions believed to be
true). Each assumption must have a unique ID and description. Insert data into id,
description fields if data exists. If missing, use  '<MISSING_Assumptions>'.",
     "7. Constraints: List all project constraints from the charter (limitations on time,
budget, resources, or technology). Each constraint must have a unique ID and
description. Insert data into id, description fields if data exists. If missing, use
'<MISSING_Constraints>'.",
     "8. Output: Combine all the previous output_result into a single JSON object
matching the template structure (fields: ProjectOverview, ProductScopeDescription,
Deliverables, ScopeInclusions, ScopeExclusions, AcceptanceCriteria, Assumptions,
Constraints). Ensure valid JSON and no additional fields."
    ]
  }
}
```


Fig 7. The instructions for Project Scope Statement

As a part of the development process, a system prompt was iteratively refined and formatted using OpenAI tools, resulting in a stable, high-level instruction set. This ensures more predictable and consistent model behavior across varying input scenarios. To reduce hallucinations, a key rule was introduced: if essential information is missing from the context, the model must leave the corresponding field blank rather than fabricate content.

This strict constraint not only improves trust in the AI-generated output but also allows for easier human validation. It encourages transparent failure handling rather than misleading completions, which is critical for professional documentation processes. As a result, the JSON files of the required documentation fragments are generated.

The examples of outputs for project scope, requirements un stakeholder register are shown in Fig. 8-10.

The solution supports structured evaluation of generated documents based on two perspectives. First, document-level criteria assess completeness, terminology, and traceability to source inputs. Second, project-specific context examines content relevance and adaptability based on the detail of provided artifacts. Testing with varied project samples confirmed that output quality improves with richer inputs. Detailed project charters produced precise and tailored documentation, while minimal inputs resulted in more generic, though structurally valid, content. This demonstrates that the system adapts to input complexity while maintaining consistent structure, making it suitable for diverse documentation scenarios across different project environments.

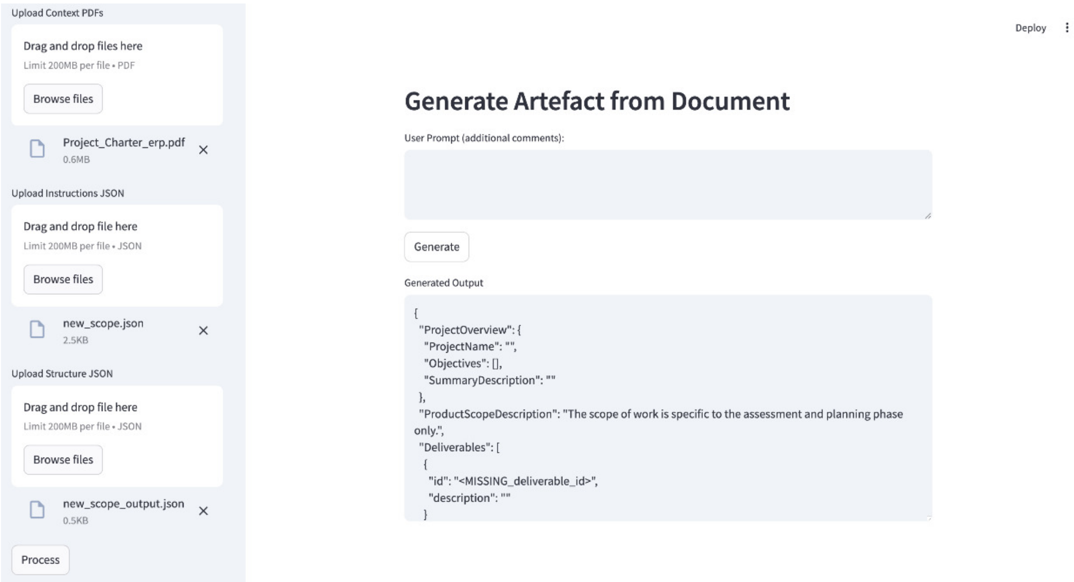

Fig 8.A fragment of output JSON for project scope

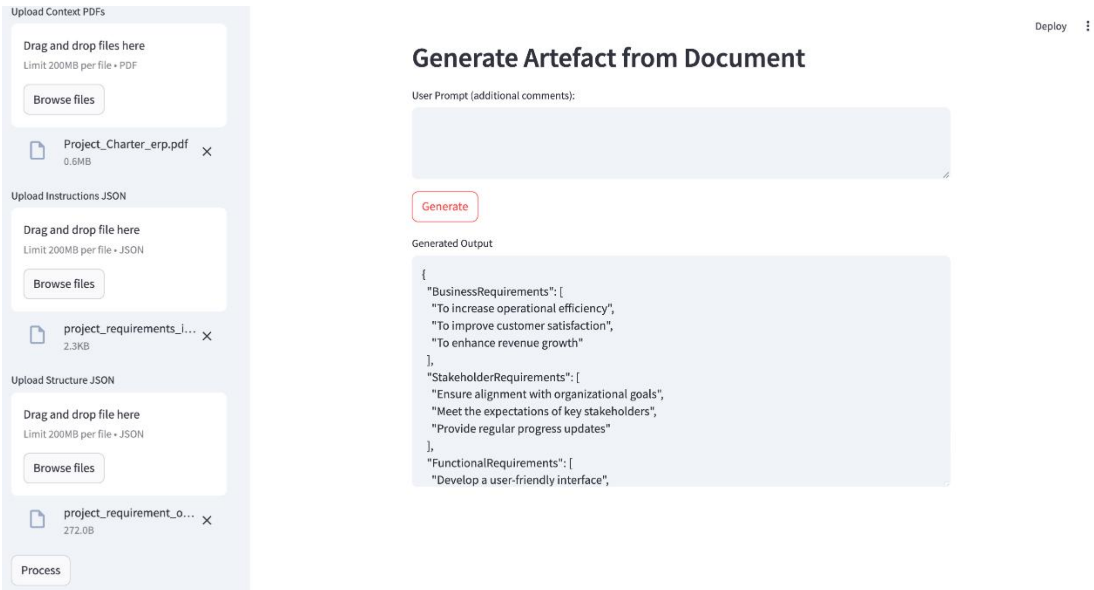

Fig 9. A fragment of output JSON for requirements

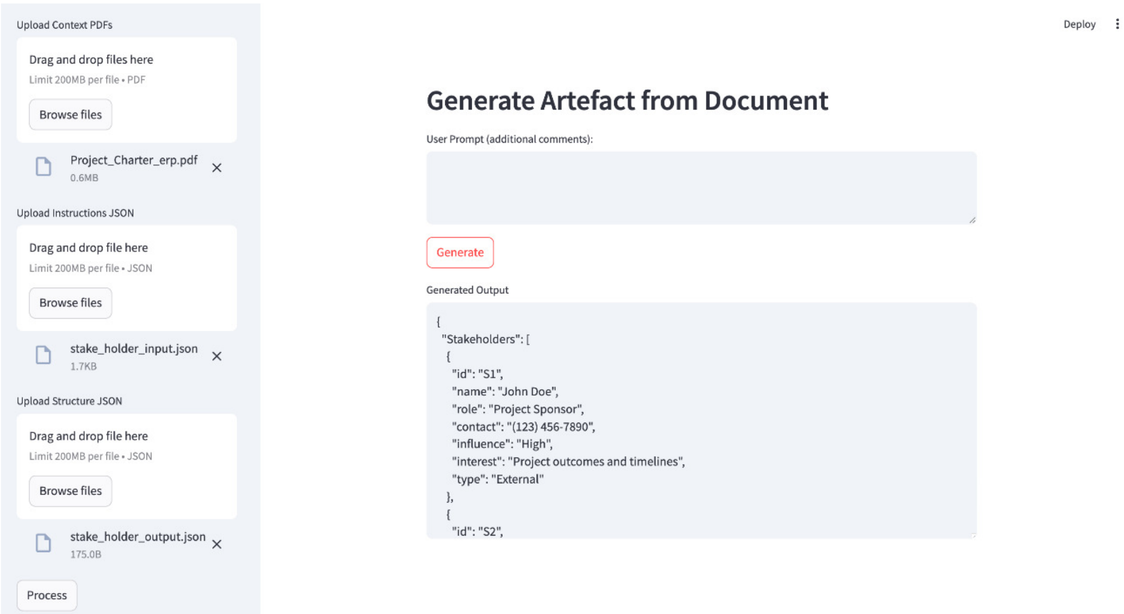

Fig 10. A fragment of output JSON for stakeholders

B. Solution Testing

The proposed approach was evaluated using two large language models (LLMs): LLaMA 3.2:3b and OpenAI GPT-4-mini. Both models received a unified instruction template and identical context limits. To ensure fairness, GPT-4-mini was restricted from using any external tools or accessing the web, relying solely on the provided input. To avoid priming effects, methodologies behind the instruction design were deliberately excluded from the prompt.

Each model processed 200-token segments with 50-token overlaps. For each evaluation step, five context fragments were drawn from the source document to ensure relevant yet limited context. Every instruction tied to an artifact was executed ten times, and outputs were stored for later analysis. To preserve neutrality, LLaMA's vector database was refreshed after each run, while GPT-4-mini was instructed to disregard prior outputs and treat each prompt as new.

Model performance was assessed using a rubric-based method, evaluating outputs across five dimensions. Each dimension was rated on a scale from 0.00 to 1.00, as detailed in Table 1.

TABLE I.
RUBRIC SCORE SCALE

Score	Rubric Criteria
0.00	Output contains incorrect or fabricated data
0.25	Output is mostly empty or contains only fragments
0.50	Output contains at least one correct element per section
0.75	Output aligns with dimensional expectations but lacks details
1.00	Output fully meets dimensional expectations

Each dimension was further weighted according to its significance in the evaluation, with the highest weights assigned to hallucination prevention and instruction compliance, given their criticality in constrained local environments [43]. Output clarity and consistency were assigned lower weights due to their less critical impact. The dimension weights are shown in Table 2.

TABLE II.
EVALUATION DIMENSIONS AND WEIGHTS

Evaluation Dimension	Weight
Data Hallucinations	0.30
Contextual Relevance	0.20
Output Consistency	0.10
Instruction Compliance	0.30
Output Clarity	0.10

A similar dimensional evaluation approach has been observed in empirical research investigating hallucinations across popular large language models. One such study assessed the hallucination rates of various LLMs by processing 1,000 documents, with results reported by [44]. Notably, the LLaMA 3.2:3b model demonstrated a hallucination rate of 7.9% in this evaluation.

In this study, simulation results were aggregated by assigning weights to each evaluation dimension, calculating average scores for each output fragment. The outcomes of the ten repeated iterations per instruction were averaged, and the dimension-specific results are presented in Table 3. Despite architectural and operational differences, that is, LLaMA 3.2:3b running locally and GPT-4-mini hosted via cloud services, both models produced comparable top results.

TABLE III.
EVALUATION DIMENSIONS AND WEIGHTS (1- HALLUCINATIONS, 2 - CONTEXT USE, 3 - CONSISTENCY , 4 - INSTRUCTION COMPLIANCE, 5 – CLARITY)

Artifact	Model	1	2	3	4	5	Overall	Scenario
Scope Statement	LLaMA 3.2:3b	0.7	0.5	0.45	0.6	0.75	0.61	One Washington
	GPT-4-mini	0.9	0.75	0.8	0.8	1.0	0.84	
Requirements Document	LLaMA 3.2:3b	0.6	0.45	0.35	0.5	0.75	0.53	
	GPT-4-mini	0.8	0.75	0.8	0.8	1.0	0.81	
Work Breakdown Structure	LLaMA 3.2:3b	0.5	0.4	0.3	0.35	0.5	0.42	
	GPT-4-mini	0.8	0.75	0.65	0.7	1.0	0.77	
Stakeholder Register	LLaMA 3.2:3b	0.5	0.5	0.3	0.4	0.5	0.45	
	GPT-4-mini	0.7	0.7	0.65	0.7	1.0	0.725	
Scope Statement	LLaMA 3.2:3b	0.5	0.5	0.45	0.4	0.75	0.49	Work Reports
	GPT-4-mini	0.8	0.75	0.8	0.6	1.0	0.75	
Requirements Document	LLaMA 3.2:3b	0.5	0.25	0.35	0.3	0.7	0.4	
	GPT-4-mini	0.7	0.75	0.8	0.6	1.0	0.71	
Work Breakdown Structure	LLaMA 3.2:3b	0.4	0.2	0.3	0.25	0.3	0.3	
	GPT-4-mini	0.7	0.6	0.65	0.5	1.0	0.64	
Stakeholder Register	LLaMA 3.2:3b	0.35	0.35	0.3	0.3	0.4	0.34	
	GPT-4-mini	0.6	0.5	0.65	0.5	1.0	0.59	
Scope Statement	LLaMA 3.2:3b	0.6	0.5	0.45	0.5	0.75	0.55	Average
	GPT-4-mini	0.85	0.75	0.8	0.7	1.0	0.79	
Requirements Document	LLaMA 3.2:3b	0.55	0.35	0.35	0.4	0.725	0.46	
	GPT-4-mini	0.75	0.75	0.8	0.7	1.0	0.76	
Work Breakdown Structure	LLaMA 3.2:3b	0.45	0.3	0.3	0.3	0.4	0.36	
	GPT-4-mini	0.75	0.68	0.65	0.6	1.0	0.7	
Stakeholder Register	LLaMA 3.2:3b	0.43	0.43	0.3	0.35	0.45	0.39	
	GPT-4-mini	0.65	0.6	0.65	0.6	1.0	0.66	

The local model performed well when processing fragments closely matching the prompt, but it struggled with semantically diverse expressions. For instance, in requirement analysis or stakeholder registers, the instructions required extracting detailed information implied in the text. Here, GPT-4-mini showed superior performance in delivering more consistent and contextually aligned outputs. A notable gap was observed between the models' ability to interpret instructions, particularly in the LLaMA model. For example, when processing a scope statement, LLaMA inferred a resource planning system implementation based solely on the vague reference to a "system."

An analysis of the results revealed that both models performed better on the One Washington sample. This document was more structured, contained instruction-aligned headings, and used less academic language. These factors are beneficial especially for the local model.

In contrast, higher hallucination rates were observed when processing the productivity project charter. For instance, the scope description included a false claim, whereas the delivery items correctly referenced the convening of a steering committee. Similarly, the "Out of Scope" section was accurate. However, in the assumptions section expected to list six elements the model included only two, of which only one was correct. This may be attributed to the token limitation, which possibly resulted in incomplete sentence inputs that the model extrapolated upon.

Evaluation of GPT-4-mini's results showed tendencies toward verbosity, as the model often expanded text to produce grammatically complete phrases. While linguistically correct, these expansions altered the intended meaning, thus reducing factual accuracy. For example, among the three output goals, only the first two were valid; the third, referencing real-time reporting, was not present in the source text. Similar inconsistencies and hallucinations were noted across other sections, including deliverables and assumptions.

C. Solution Drawback and Improvements

Another key observation during evaluation was the difference in how the models responded to insufficient or ambiguous information. LLaMA 3.2:3b tended to return empty or partial outputs when it could not confidently locate the requested data. In contrast, GPT-4-mini frequently attempted to complete outputs regardless, even when context was lacking. Often generating fabricated content in direct violation of the provided instructions.

In the context of this study, where data integrity is of high importance, LLaMA's behavior was actually preferable. Empty outputs are easier to verify, as they indicate uncertainty rather than assumed correctness, which reduces the risk of unintentionally integrating hallucinated data into downstream processes.

The results suggest that smaller, locally hosted models like LLaMA 3.2:3b, when given clearly defined instructions and constrained environments, are capable of producing viable outputs, particularly when a human reviewer is involved in validation. Although GPT-4-mini demonstrated superior generalization abilities, its tendency to confidently fabricate data limited its advantage in this controlled use case.

The test design used small input samples and short text fragments. In several cases, both models failed to retrieve relevant content and instead included false or unrelated claims. LLaMA also demonstrated high variability between runs: in multiple cycles, it returned empty or low-information responses, likely due to how fragments were selected via the nearest-neighbor search over the vector database using Ollama embeddings.

Each processing step queried only five top-matching fragments, assuming semantic accuracy in the embedding model. However, no additional validation of the relevance of retrieved fragments was performed. Furthermore, the model's iterative process continuously appended data to key/value stores without restriction, potentially leading to overwriting or compounding errors in later stages.

In summary, while GPT-4-mini showed superior fluency and semantic approximation, it introduced more hallucinations than expected. LLaMA 3.2:3b, though less consistent, adhered more strictly to constraints and demonstrated a conservative generation strategy more suitable for high-integrity documentation workflows in local AI-assisted systems.

VI. Conclusion

This study demonstrates that the integration of generative AI into IT project documentation processes is not only feasible but also valuable, which is provided by the frame within a model-driven and standards-aligned architecture. By combining the expressive capabilities of large language models with the structural rigor of model-driven development, the proposed solution supports the generation of high-quality, semantically consistent documentation artifacts during the early phases of IT projects.

From a theoretical perspective, this work contributes to the design science research body of knowledge by introducing a novel artifact: a model-based framework that constrains and guides generative AI outputs. It builds upon the principles of semantic control in AI-assisted software engineering, showing that MDD structures can serve as effective scaffolds for LLM behavior. Additionally, it expands on prior work in documentation automation by demonstrating how prompt engineering, semantic traceability, and formal process alignment can mitigate common challenges such as hallucinations and incoherence.

In terms of practical implications, the solution enables project teams, especially in resource-constrained environments, to accelerate and standardize documentation workflows without compromising methodological integrity. The prototype shows that even with local infrastructure, it is possible to harness generative AI tools securely and effectively. Reusable, cross-method instruction sets and alignment with common standards further enhance the applicability and scalability of the approach in varied project contexts.

Several key findings of the research are as follows:

1. Structured inputs and well-defined transformation rules significantly improve the quality and reliability of AI-generated documents.
2. Embedding documentation within a model-driven framework allows for semantic validation, traceability, and reduced ambiguity.
3. Local deployment of LLM-based tools is technically feasible and aligns with data governance requirements in many organizations.
4. Standard elements (e.g., objectives, risks, assumptions) can be reused across methods, enabling cross-framework consistency in documentation practices.

This research opens new avenues for extending the framework to other documentation types and domains (e.g., test plans, architecture overviews, compliance checklists), and for integrating dynamic feedback loops to iteratively refine generated content based on stakeholder input or project evolution.

Future work could explore as follows:

1. Comparative evaluation across multiple LLMs and domain contexts;
2. Deeper integration with enterprise modeling tools or agile project platforms;
3. Automated validation mechanisms for AI-generated artifacts.

In conclusion, the results affirm that generative AI, when properly constrained and guided, can enhance formal documentation practices in IT project management. Rather than replacing human expertise, such tools augment human capabilities, i.e., improving efficiency, consistency, and knowledge transfer in complex project environments.

References

[1] X. Zhang and B. Dorn, "Accelerating software development through agile practices: A case study of a small-scale, time-intensive web development project at a college-level IT competition," *J. Inf. Technol. Educ.*, vol. 11, pp. 1–12, 2012, doi: 10.28945/1545.

[2] H. Kerzner, "Project Management: A Systems Approach to Planning, Scheduling, and Controlling", 13th ed. Hoboken, NJ, USA: Wiley, Mar. 2022, ISBN 978-1119805373.

[3] B. Habib and R. Romli, "A Systematic Mapping Study on Issues and Importance of Documentation in Agile," 2021 IEEE 12th International Conference on Software Engineering and Service Science (ICSESS), Beijing, China, 2021, pp. 198-202, doi: 10.1109/ICSESS52187.2021.9522254

[4] W. Behutiye, P. Seppänen, P. Rodríguez, and M. Oivo, "Documentation of quality requirements in agile software development," in Evaluation and Assessment in Software Engineering (EASE' 20), Trondheim, Norway, Apr. 15–17, 2020, ACM, New York, NY, USA, pp. 1–10, doi: 10.1145/3383219.3383245

[5] A. Ataman, "Data quality in AI: Challenges, importance & best practices," AIMultiple. [Online]. Available: https://research.aimultiple.com/data-quality-ai/

[6] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, et al., "Language models are few-shot learners," Adv. Neural Inf. Process. Syst., vol. 33, pp. 1877–1901, 2020, doi: 10.48550/arXiv.2005.14165

[1] P. Liang, R. Bommasani, D. Tsipras, et al., “Holistic evaluation of language models,” arXiv preprint arXiv:2211.09110, 2022, doi: 10.48550/arXiv.2211.09110.
[2] OpenAI, “GPT-4 Technical Report,” OpenAI, Mar. 2023. [Online]. Available: https://openai.com/research/gpt-4
[3] F. Nafz, M. Krajinovic, and M. Ley, “Artificial intelligence in software documentation: Embracing the documentation as code paradigm,” in “Go Where the Bugs Are”, G. Ernst et al., Eds., “Lecture Notes in Computer Science”, vol. 15765, Cham, Switzerland: Springer, 2025, doi: 10.1007/978-3-031-92196-4_14.
[4] K. J. Ajeigbe and O. Emma, “Dynamic Documentation Generation with AI,” 2024. Available at: https://www.researchgate.net/publication/390265865_Dynamic_Documentation_Generation_with_AI
[5] K. Dearstyne, A. Rodriguez, and J. Cleland-Huang, “Supporting Software Maintenance with Dynamically Generated Document Hierarchies,” arXiv preprint arXiv:2408.05829, Aug. 2024, doi: 10.48550/arXiv.2408.05829
[6] S. Mehta, A. Rogers, and T. Gilbert, “Dynamic Documentation for AI Systems,” arXiv preprint arXiv:2303.10854, Mar. 2023, doi: 10.48550/arXiv.2303.10854.
[7] C. Leyh, A. Lorenz, M. J. Faruga, and L. Koller, “Critical Success Factors for ERP Projects Revisited: An Update of Literature Reviews,” in Proc. 19th Conf. on Computer Science and Intelligence Systems (FedCSIS), vol. 39, Annals of Computer Science and Information Systems, pp. 131–140, 2024, doi: 10.15439/2024F6271
[8] T. Vetriselvi, M. Mathur, and M. Bhuvaneswari, “Applying Generative AI to Create SOP, Reducing API Costs Through Prompt Compression and Evaluating LLM Responses with Tonic Validate RAG Metrics,” in Proc. 4th Int. Conf. Ubiquitous Comput. Intell. Inf. Syst. (ICUIS), 2024, doi: 10.1109/ICUIS64676.2024.10867024
[9] S. Tsuchiwata, Y. Tanabe, Y. Hosoya, and K. Aoyama, “Generative AI Driven Clinical Drug Development,” Japanese Journal of Clinical Pharmacology and Therapeutics, 2025, doi: 10.3999/jscpt.56.2_109
[10] A. Barcaui and A. Monat, “Who is better in project planning? Generative artificial intelligence or project managers?,” Project Leadership and Society, vol. 4, 2023. 10.1016/j.plas.2023.100101
[11] A. Bahi, J. Gharib, and Y. Gahi, “Integrating Generative AI for Advancing Agile Software Development and Mitigating Project Management Challenges,” Int. J. Adv. Comput. Sci. Appl., vol. 15, no. 1, 2024, doi: 10.14569/IJACSA.2024.0150306
[12] Z. Alliata, T. Singhal, and A.-M. Bozagiu, “The AI Scrum Master: Using Large Language Models (LLMs) to Automate Agile Project Management Tasks,” in Lect. Notes Bus. Inf. Process., 2025, doi: 10.1007/978-3-031-72781-8_12
[13] M. AbuAlfateh, M. Ali, and M. Messaadia, “Establishing Key Performance Indicators for Human-Generative AI Collaboration,” Studies in Big Data, vol. 110, 2025, doi: 10.1007/978-3-031-83911-5_42
[14] V. Bilgram and F. Laarmann, “Accelerating Innovation With Generative AI: AI-Augmented Digital Prototyping and Innovation Methods,” IEEE Eng. Manag. Rev., vol. 51, no. 2, pp. 40–47, 2023, doi: 10.1109/EMR.2023.3272799
[15] N. Ibadildin, Z. Kenzhin, G. Yeshenkulova, and A. Kadyrova, “Artificial Intelligence in Project Management: A Bibliometric Analysis,” Problems and Perspectives in Management, vol. 23, no. 1, pp. 130–142, 2025, doi: 10.21511/ppm.23(2).2025.17
[16] S. Salimimoghadam, A. N. Ghanbaripour, R. J. Tumpa, and M. Skitmore, “The Rise of Artificial Intelligence in Project Management: A Systematic Literature Review of Current Opportunities, Enablers, and Barriers,” Buildings, vol. 15, no. 2, 2025. 10.3390/buildings15071130
[17] O. Nikiforova, J. Grabis, O. Pastor, K. Babris, M. K. Miļūne, and R. Bobkovs, “Model-Based Methodology for Development of IT Project Management Plan and Scope Using Artificial Intelligence: Project in Progress,” in Proc. 19th Int. Conf. on Research Challenges in Information Science (RCIS), CEUR Workshop Proc., 2025 (in press).
[18] J. White, C. Kirchner, T. Paschal, S. Hays, A. Kazerouni, T. Mytkowicz, and M. Monperrus, “A Prompt Pattern Catalog to Enhance Prompt Engineering with ChatGPT,” arXiv preprint arXiv:2302.11382, 2023, doi: 10.48550/arXiv.2302.11382
[19] P. Liu, W. Yuan, J. Fu, Z. Jiang, H. Hayashi, and G. Neubig, “Pre-train, Prompt, and Predict: A Systematic Survey of Prompting Methods in Natural Language Processing,” ACM Comput. Surv., vol. 55, no. 9, Art. no. 195, pp. 1–35, Sep. 2023, doi: 10.1145/3560815
[20] O. Nikiforova, K. Babris, U. Karlovs-Karlovskis, M. Narigina, A. Romanovs, A. Jansone, J. Grabis, and O. Pastor, “Model Transformations Used in IT Project Initial Phases: Systematic Literature Review,” Computers, vol. 14, no. 2, p. 40, 2025, doi: 10.3390/computers14020040
[21] O. Nikiforova, K. Babris, M. K. Miļūne, N. Tanguturi, and Ó. Pastor, “Key Artefacts in the Initial Phases of IT Project Management: Systematic Mapping Study,” in Proc. 20th Int. Conf. Evaluation of Novel Approaches to Software Engineering (ENASE), SciTePress, 2025, pp. 773–781, doi: 10.5220/0013471000003928
[22] A. R. Hevner, S. T. March, J. Park, and S. Ram, “Design Science in Information Systems Research,” *MIS Quarterly*, vol. 28, no. 1, pp. 75–105, 2004, doi: 10.2307/25148625.
[23] M. Brambilla, J. Cabot, and M. Wimmer, Model-Driven Software Engineering in Practice, 2nd ed. Cham, Switzerland: Springer, 2017.
[24] O. Pastor and J. Molina, Model-Driven Architecture in Practice: A Software Production Environment Based on Conceptual Modeling. Berlin/Heidelberg, Germany: Springer, 2007. [Online]. Available: https://doi.org/10.1007/978-3-540-71868-0
[25] O. Nikiforova, M. K. Miļūne, K. Babris, and O. Pastor, “Generation of IT Project Documentation Elements from a Model Transformation Chain,” in Proc. Int. Conf. on Software Technologies (ICSOFT), 2025, pp. 1–10, DOI: 10.5220/0013568300003964
[26] Agile Manifesto (2001) https://agilemanifesto.org/
[27] ISO/IEC/IEEE, Systems and software engineering — Life cycle processes — Requirements engineering, ISO/IEC/IEEE 29148:2018, International Organization for Standardization, Geneva, Switzerland, 2018
[28] ISO/IEC/IEEE, Systems and software engineering — Life cycle processes — Project management, ISO/IEC/IEEE 16326:2019, International Organization for Standardization, Geneva, Switzerland, 2019
[29] ISO, Project, programme and portfolio management — Guidance on project management, ISO 21502:2020, International Organization for Standardization, Geneva, Switzerland, 2020.
[30] P. Bourque and R. E. Fairley, Eds., Guide to the Software Engineering Body of Knowledge (SWEBOK Guide), Version 4.0, IEEE Computer Society, 2024. [Online]. Available: https://www.computer.org/education/bodies-of-knowledge/software-engineering
[31] Project Management Institute, A Guide to the Project Management Body of Knowledge (PMBOK® Guide), 7th ed., Newtown Square, PA, USA: PMI, 2021
[32] O. Nikiforova, M. Kirikova, and N. Pavlova, “Two-hemisphere driven approach: Application for knowledge modeling,” in Proc. 7th Int. Baltic Conf. Databases and Information Systems, 2006, pp. 244–250, art. no. 1678503. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-34250753483
[33] O. Nikiforova and K. Gusarovs, “Comparison of BrainTool to other UML modeling and model transformation tools”, AIP Conf. Proc., vol. 1863, art. no. 330005, 2017. doi: 10.1063/1.4992503
[34] J. He, M. Rungta, D. Koleczek, A. Sekhon, F. X. Wang, and S. Hasan, “Does Prompt Formatting Have Any Impact on LLM Performance?,” arXiv preprint arXiv:2411.10541, 2024, doi: 10.48550/arXiv.2411.10541
[35] A. Madaan, N. Tandon, P. Gupta, S. Hallinan, L. Gao, S. Wiegreffe, et al., “Self-Refine: Iterative Refinement with Self-Feedback,” arXiv preprint arXiv:2303.17651, 2023, doi: 10.48550/arXiv.2303.17651
[36] OpenAI, ChatGPT (May 2024 version) [Large Language Model]. Available: https://chat.openai.com/
[37] T. Bao, J. Yang, Y. Yang, and Y. Yin, "RM2Doc: A tool for automatic generation of requirements documents from requirements models," in Proc. ACM/IEEE 44th Int. Conf. Softw. Eng.: Companion Proc. (ICSE ’22), New York, NY, USA, 2022, pp. 188–192, doi: 10.1145/3510454.3516850.
[38] Vectara. (2024). Hallucination leaderboard [Computer software]. GitHub. https://github.com/vectara/hallucination-leaderboard