Prompt-Based Automation of Agile Requirements Documentation: Design for Limited-Infrastructure Environment

https://doi.org/10.31713/MCIT.2025.032

Roman Kruk
The Department of Computer Science and Applied Mathematics
The National University of Water and Environmental Engineering, NUWEE
Rivne, Ukraine
r.a.kruk@nuwm.edu.ua

Nataliia Zhukovska
The Department of Computer Science and Applied Mathematics
The National University of Water and Environmental Engineering, NUWEE
Rivne, Ukraine
n.a.zhukovska@nuwm.edu.ua

Abstract—**This study presents a proof-of-concept design and evaluation of a prompt-based automation approach for Agile requirements documentation in a limited-infrastructure environment. Building upon a previously proposed five-step LLM-based analytical framework, the research explores how structured prompt engineering can replicate the behavior of a full middleware system for transforming Jira exports into publishable Feature Specification Documents. The experiment involved 12 iterative prompt-refinement cycles applied to real project data. The final Gem's configuration achieved 79% successful cases (improved by 27% when project knowledge artifacts were included). The results confirm that prompt-driven automation can effectively bridge unstructured Agile artifacts and formal specification, offering a lightweight and auditable alternative to traditional requirements documentation workflows.**

Keywords — Agile, Requirements Engineering; Prompt Engineering; Large Language Models; Feature Specification; Process Automation

I. Introduction

The study addresses persistent challenges in Agile Requirements Engineering (RE), where semi-structured artifacts such as user stories and acceptance criteria often lack consistency, traceability, and integration with formal system documentation[1]. In contemporary IT practice, especially in large Agile teams, documentation quality remains a key bottleneck that affects project predictability and validation[2].

The object of research is the process of requirements analysis and transformation in Agile projects; the subject is the application of generative AI through structured prompt engineering to automate and improve that process.

The goal of this research stage is to evaluate how the previously proposed five-step LLM-based method for requirements analysis can be adapted into a prompt-only proof-of-concept (PoC) within a limited project infrastructure.

II. Context of the Study

This work continues a multi-stage research program initiated in 2023–2025, where earlier studies developed:

1. a conceptual data-flow model of Agile requirements and its bottlenecks [3];

2. the intermediary LLM-based middleware integrated with Jira and Neo4j for graph-based knowledge modeling[4]; and

3. validation of a five-step analytical framework achieving high correctness results during the first test run[5].

At the current stage, the focus shifts from full middleware deployment to manual PoC implementation within a real IT project at SoftServe Inc., constrained to only Jira, Confluence, and Gemini tools.
The study examines how the core method can operate in such low-infrastructure environments and whether carefully designed prompts (custom Gems) can reproduce the essential analytical behavior of the full system.

III. Methodology

An iterative experimental design was used, consisting of 12 consecutive prompt engineering cycles applied to real Jira epics exported from a SoftServe project. Each iteration processed 50 epics using a modified five-step requirements analysis logic.
The choice to focus on PoC implementation rather than the full middleware architecture was driven by two factors:

1. the ability to test the approach on a live corporate project under restricted infrastructure (Jira, Confluence, Gemini only), and

2. the need to validate the scientific novelty of the five-step analytical method itself, independent of system integration.

Two custom Gems were configured in Gemini: one for transforming Jira exports into Feature Specification Documents (FSD) and another for dependency analysis.

In the scope of this paper only the first prompts will be presented and described.

Custom Gems allow prompt persistence (multi-use configuration) and inclusion of knowledge artifacts — project structure, terminology, and glossary — which improve contextual accuracy. The evaluation of generated documentation was based on three quality criteria:

- Relevance - measures how well the generated Feature Specification reflects the actual content and intent of the original Jira requirements. A highly relevant document preserves business meaning, scope, and context without introducing unrelated information.
- Completeness - evaluates whether all important requirement elements (user stories, acceptance criteria, dependencies, and constraints) were captured and represented in the output. A complete document should cover every Jira item without omissions.
- Correctness - evaluates the factual and logical accuracy of the generated specification. It verifies that requirements are expressed clearly, testably, and without semantic distortion or contradictions compared to the source data.

Each scored from 1 to 5. A case was marked successful if it achieved at least 11 total points and no score below 3. This scoring framework was applied consistently across all 12 experimental iterations.

IV. Study Results

Twelve experimental iterations were conducted, each involving prompt refinement and evaluation over 50 epics. The overall trend showed progressive improvement in output quality, with the twelfth prompt achieving the highest performance (Table 1).

TABLE I. Success Results of the Iterative ‘FeatureSpecificationWriter’ Prompt Engineering

Itera tion	Version Name	Successful Cases (%)
1	Baseline simple prompt	34
2	Added structure outline	39
3	Added FR/NFR/TR separation	44
4	Added acceptance criteria preservation	48
5	Introduced dependency section	55
6	Added risk & constraint handling	48
7	Introduced traceability table	51
8	Rewritten to improve clarity & grouping	62
9	Added normalization & clustering logic	64
10	Included Gherkin[6][7] transformation rules	67
11	Optimized contextual reasoning (project structure & overview artefacts)	72
12	Glossary knowledge artefact included (final Feature SpecificationWriter)	79

A. *Design of the FeatureSpecificationWriter Gem*

The FeatureSpecificationWriter (FSW) Gem serves as a reusable generative configuration designed to transform Jira-exported requirements into a clean, publishable Feature Specification Document (FSD) suitable for direct insertion into documentation systems such as Confluence or SharePoint. Its purpose is to consolidate semi-structured Agile artifacts — stories, tasks, and acceptance criteria — into a unified and formally structured specification, ensuring completeness, clarity, and traceability without the need for middleware or database integration.

The Gem operates entirely through prompt logic and controlled instruction chaining[8][9]. It replicates the analytical reasoning of the five-step requirements analysis framework proposed in previous research—clustering, linking, dependency detection, deduplication, and traceability—within a single self-contained generative process. The FSW receives one or several Jira CSV exports as input, optionally accompanied by project briefs, diagrams, or glossary files. The output is a markdown-structured FSD ready for publication. Unlike ordinary prompts that rely on transient context, the Gem configuration ensures persistence and reproducibility: it can be reused across projects while retaining the same logical behavior and section structure.

The process begins with input normalization. The Gem has a command to automatically detects field names and separators, handles quoted newlines, trims whitespace, and maps variable column labels such as Summary, Title, Description, or Acceptance Criteria to canonical categories. If any essential field is missing or ambiguously labeled, the Gem requests a header sample or infers mappings from semantic similarity. This preprocessing step guarantees compatibility across heterogeneous Jira export formats.

Next, the grouping and deduplication phase organizes tickets by their Epic link or component reference. Duplicate or fragmented tickets sharing the same summary within one epic are merged, while each issue still retains a unique traceability reference.

The content extraction and rewriting stage reads all relevant text from the Description, Acceptance Criteria, and Comments fields, preserving their full meaning while reformulating them into precise, testable requirement statements. Any uncertain or incomplete information is flagged as TBD or Open Question rather than being omitted. The Gem never introduces new facts not present in the source material, thereby minimizing hallucinations and ensuring factual fidelity.

Requirements are then classified into four categories: User-Level Requirements (ULRs) describing business capabilities in concise, outcome-oriented form; Functional Requirements (FRs) detailing system behavior, inputs, outputs, and rules; Non-Functional Requirements (NFRs) capturing performance, security, usability, and reliability constraints; and Transitional Requirements (TRs) specifying migration, rollout, and deprecation activities.

The FSW also extracts technical design notes — non-requirement information such as architectural constraints, API endpoints, or schema references — from ticket comments. These notes provide valuable engineering context but remain distinct from normative requirements. The Gem further identifies dependencies between functional units by detecting shared entities, data sources, or API references, and records them in a dedicated section.

Each generated requirement receives a stable identifier based on its sequential position within the document. These identifiers feed into a traceability matrix mapping each requirement to its original Jira key. This matrix guarantees full coverage: every Jira ticket must appear at least once in the matrix or in the appendix.

The resulting output follows a consistent and document-friendly structure: feature overview, user-level requirements, functional requirements (with acceptance criteria and Gherkin examples), non-functional requirements, transitional requirements, technical design notes, dependencies, risks and constraints, traceability matrix, open questions and appendix.

This stable format enables instant publication and supports automated cross-linking with other documentation artifacts. Within Gemini, the FSW Gem may include adjustable parameters such as granularity (one document per epic or multi-epic aggregation), verbosity (standard or detailed), Gherkin conversion mode, and output syntax.

A key strength of the FSW Gem lies in its anti-loss design. Even poorly structured Jira tickets are represented: missing data trigger placeholders rather than omissions, and all unclassified content is archived in the Appendix. This mechanism substantially reduces information loss compared to typical single-pass summarization prompts. Empirical testing has shown that hallucinations are nearly eliminated, while the primary failure mode remains the occasional omission of low-context requirements.

B. FeatureSpecificationWriter Gem usage on Agile Project

The FeatureSpecificationWriter Gem was applied and is recommended to be applied within an Agile project that has limited automation resources for requirements engineering. Figure 1 depicts how a human Agent collaborates with the Issue Management System (Jira), the Gem, and the Document Management System (DMS) to transform exported Agile artifacts into structured knowledge.

The process begins when the Agent initiates the export of an epic or a set of issues from Jira. This step corresponds to obtaining semi-structured requirement data in CSV or similar textual form. Once the export is complete, the Agent sends this data to the Gem, which acts as the intelligent processing core.

Inside the Gem, the FeatureSpecificationWriter executes its structured reasoning chain: it parses the exported dataset, normalizes fields, groups tickets by epic or component, rewrites user stories into formalized requirements, and assembles a complete Feature Specification Document (FSD).

After processing, the Gem returns the generated FSD to the Agent. This output is immediately suitable for documentation platforms such as Confluence or SharePoint. The Agent then uploads the new specification to the Document Management System, effectively enriching it with updated project knowledge. The DMS acknowledges a successful update, confirming that the new knowledge — derived from Jira through the Gem — has been integrated into the project's documentation repository.

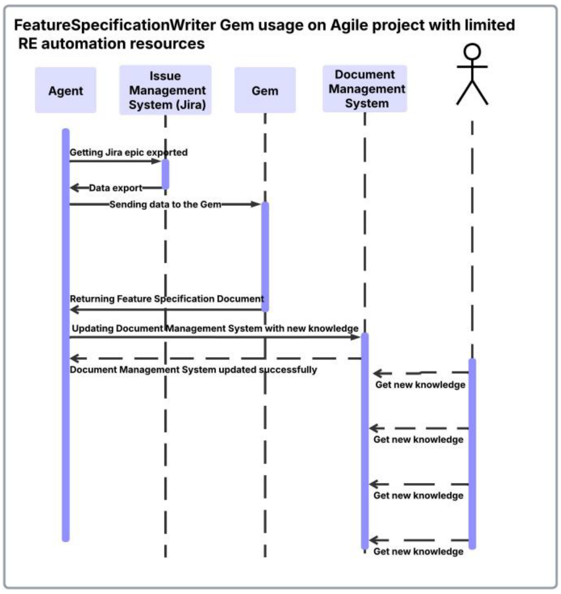
FeatureSpecificationWriter Gem usage on Agile project with limited RE automation resources
Agent
Issue Management System (Jira)
Gem
Document Management System
Getting Jira epic exported
Data export
Sending data to the Gem
Returning Feature Specification Document
Updating Document Management System with new knowledge
Document Management System updated successfully
Get new knowledge
Get new knowledge
Get new knowledge
Get new knowledge

Figure 1. Sequence diagram on the FeatureSpecificationWriter Gem usage on Agile project with limited RE automation resources.

From this point onward, the user (stakeholder) interacts with the DMS to access or query this new knowledge.

V. Advantages and Constraints

A. Advantages

A primary advantage of the Gem lies in its ability to replicate a full requirements engineering pipeline within a single generative operation. This makes it especially valuable in infrastructure-limited environments, where access to graph databases or REST APIs is restricted, yet project teams still require consistent and audit-ready documentation.

Another major benefit is reproducibility. Because the Gem is configured as a reusable generative module with persistent system logic, it can be applied across projects with minimal prompt modification.

The Gem also demonstrates high interpretability and transparency compared to conventional LLM usage. By avoiding black-box automation layers, analysts can observe each conversion stage and validate results interactively.

A further advantage is contextual scalability. The inclusion of knowledge artifacts, such as project glossaries, structural ontologies, or domain context significantly enhances precision. Empirical evaluation revealed an average improvement of about 27% in document accuracy and completeness when such contextual data were preloaded into the Gem.

Lastly, the approach reduces manual workload and cognitive fatigue for business analysts.

B. Constraints

Despite its strengths, the Gem faces notable limitations. The most significant issue observed during experimentation is partial data omission: certain low-context or ambiguously worded Jira tickets may not be fully captured in the generated output. While hallucinations were almost entirely absent due to the strict "no data invention" rule, the model occasionally skips minor requirements that lack explicit acceptance criteria or functional detail. This makes post-generation review essential for ensuring completeness.

Another constraint involves manual effort in data preparation and evaluation. Because the Gem relies on exported Jira files rather than direct API integration, analysts must perform CSV normalization and upload manually.

Performance is also influenced by prompt sensitivity. Minor variations in phrasing, order, or emphasis within the system prompt may affect output consistency. Maintaining stability across Gem versions requires careful version control and continuous prompt engineering refinement.

Lastly, while contextual grounding improves output quality, it also raises dependency on knowledge artifacts. Outdated or incomplete project glossaries may bias the Gem's interpretation, resulting in incorrect term mappings or missed dependencies.

C. Novelty Compared to Classical Agile RE

The application of the FeatureSpecificationWriter Gem represents a paradigm shift from conventional Agile documentation practices, which typically depend on human interpretation and loosely structured user stories. In standard Agile workflows, documentation evolves organically through tickets, often leading to inconsistencies, duplication, and weak traceability. The Gem introduces a formalized, LLM-driven synthesis layer that consolidates all relevant artifacts into a single, structured knowledge representation while preserving their semantic integrity.

Moreover, the Gem's modular architecture, based entirely on prompt logic rather than code, makes it adaptable and transparent, allowing analysts to fine-tune reasoning steps without modifying system infrastructure.

VI. CONCLUSIONS

This research stage demonstrated the technical feasibility of applying the five-step LLM-based requirements analysis method through prompt engineering alone, without middleware integration.
By configuring reusable Gemini Gems with embedded project knowledge and a structured Feature Specification prompt, analysts achieved over 70% document correctness and completeness under limited infrastructure conditions.

In summary, the Gem serves as a semantic bridge between unstructured issue-tracking data and structured documentation. It enables manual yet repeatable automation: the Agent provides input and validation, Jira serves as the data source, the Gem performs generative reasoning, and the DMS becomes the persistent knowledge base. Even in the absence of full middleware or graph-database integration, this lightweight workflow ensures continuous transformation of Agile artifacts into formalized, accessible, and reusable project knowledge.

Future work will focus on engineering an advanced Gem capable of autonomously analyzing and maintaining internal knowledge of functional and data dependencies between system components — operating fully locally, without relying on external databases or persistent storage.

This will complete the transition from static prompt templates to adaptive, context-aware AI assistants for requirements management.

ACKNOWLEDGMENT

The author expresses sincere gratitude to SoftServe Inc. for providing access to real Agile project environments, infrastructure, and professional expertise that made this research possible.

REFERENCES

[1] Hoy Z., Xu M. Agile software requirements engineering challenges-solutions–a conceptual framework from systematic literature review. Information. 2023. Vol. 14, no. 6. P. 322. URL: https://doi.org/10.3390/info14060322 (date of access: 29.09.2025).

[2] I. Inayat et al. A systematic literature review on agile requirements engineering practices and challenges. Computers in human behavior. 2015. Vol. 51. P. 915–929. URL: https://doi.org/10.1016/j.chb.2014.10.046 (date of access: 29.09.2025).

[3] Kruk R., Zhukovska N. Analysis of bottleneck points based on the software requirements data-flow model in Agile projects. Mathematical Modelling and Computing. 2025. Vol. 12, no. 2. P. 628-639. URL: https://doi.org/10.23939/mmc2025.02.628 (date of access: 04.10.2025).

[4] Kruk R., Zhukovska N. Toward AI-assisted Framework for Agile Requirement Knowledge Management: Five-Stage Generative LLM Approach. 2025 15th International Conference on Advanced Computer Information Technologies., in press.

[5] Kruk R., Zhukovska N. Five-step Semantic Analysis Middleware for Atlassian Jira Software Requirements Transformation using Generative Large Language Model. 2025: Mathematical and computer modelling. Series: Technical sciences. Vol. 27. P. 40-57. URL: https://doi.org/10.32626/2308-5916.2025-27.40-57 (date of access: 04.10.2025) [in Ukrainian].

[6] C. Lauer, C. Sippl. Benefits of Behavior Driven Development in Scenario-based Verification of Automated Driving. 2022 IEEE 25th International Conference on Intelligent Transportation Systems (ITSC). URL: https://doi.org/10.1109/ITSC55140.2022.9922498 (date of access: 29.09.2025).

[7] Nic Werner. Writing User Stories With Gherkin. URL: https://medium.com/@nic/writing-user-stories-with-gherkin-dda63461b1d2 (date of access: 29.09.2025).

[8] Leon Nichols. The Ultimate Guide to Google Gemini Gems. URL: https://leonnicholls.medium.com/the-ultimate-guide-to-google-gemini-gems-78182be784af (date of access: 29.09.2025).

[9] A. Kale et al. Unveiling the Power of AI Prompt Engineering: A Comprehensive Exploration. International Conference Electrical Energy Systems. 2024. URL: https://doi.org/10.1109/ICEES61253.2024.10776884 (date of access: 29.09.2025).