

Can LLMs Identify Missing Elements in Requirements Elicitation? An Empirical Evaluation Using Production-Derived User Stories in an Industry Setting

Vinay Kabadi 

School of Computing and Information Systems, The University of Melbourne, Australia

Abhiram Wuntakal 

Trawelli Pty Ltd, Australia

Naresh Gutha 

Industry Practitioner, Australia

Lahiri Bellarykar 

Industry Practitioner, Australia

Xuan Bach D. Le 

School of Computing and Information Systems, The University of Melbourne, Australia

Patanamon Thongtanunam 

School of Computing and Information Systems, The University of Melbourne, Australia

Christoph Treude 

School of Computing and Information Systems, Singapore Management University, Singapore

Abstract

Business Requirements often fail to fully capture the complete set of system requirements they intend to describe, resulting in under-specification and measurable gaps between documented requirements and production-ready implementations. While experienced business analysts attempt to mitigate these gaps using domain expertise, the resulting artifacts often remain incomplete.

This study uses Cross-Pollination, a domain-agnostic methodology for transforming incomplete natural language Business Requirements into comprehensive, production-grade functional user stories. To evaluate the completeness of LLM-generated user stories, the study systematically benchmarks them against realistic production artifacts. It further investigates the capability of Large Language Models (LLMs) to bridge the coverage gap between human-authored backlogs and production systems. It identifies the requirements that are overlooked during human-driven elicitation but are recovered through an LLM-assisted approach.

The cross-pollination technique is a structured multi-phase methodology supported by a gap analysis taxonomy and a domain complexity calibration mechanism to establish a minimum baseline for production readiness. The methodology is applied across real-world domains, and a comparative evaluation of analyst-authored, LLM-generated, and production-derived user stories is conducted. Empirical results demonstrate that a substantial proportion of requirement gaps can be systematically identified and addressed through this approach.

The findings indicate that Cross-Pollination provides a reproducible and scalable methodology for translating natural-language requirements into more complete and production-aligned user stories. The results demonstrate that while analysts produce precise requirements, LLM-generated user stories achieve higher recall across all evaluated domains and are effective in uncovering implicit and non-obvious requirements. These findings indicate that an integrated approach, with LLM-assisted elicitation functioning as a complementary mechanism to human expertise, produces more comprehensive and reliable requirement backlogs than either method independently.

2012 ACM Subject Classification Computing methodologies → Natural language processing; General and reference → Empirical studies; Software and its engineering → Requirements analysis; Software and its engineering → Software development methods



© Vinay Kabadi, Abhiram Wuntakal, Naresh Gutha, Lahiri Bellarykar, Xuan Bach D. Le, Patanamom Thongtanunam, and Christoph Treude;

licensed under Creative Commons License CC-BY 4.0

20th International Symposium on Empirical Software Engineering and Measurement (ESEM 2026).

Editors: Robert Feldt, Maria Paasivaara, Daniel Mendez, Stefan Wagner, and Marvin Muñoz Barón;

Article No. 82; pp. 82:1–82:20



Leibniz International Proceedings in Informatics

LIPIC Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Keywords and phrases Software Engineering, Requirement Elicitation, Large Language Model (LLM), User Stories

Digital Object Identifier 10.4230/LIPIcs.ESEM.2026.82

Category Software Engineering in Practice Track Paper

1 Introduction

Over the past decades, software engineering has evolved significantly, with recent advances in artificial intelligence, including machine learning, large language models (LLMs), generative AI, and emerging agentic systems, further accelerating this transformation. Despite these advances, successful software development continues to depend fundamentally on the clarity, completeness, and quality of requirements and user stories.

Requirements elicitation (RE) remains a critical activity in software development, as business requirements establish the connection between organisational intent and system behaviour, while user stories translate these requirements into actionable units for downstream activities such as design, development, testing, and maintenance. Although requirements are typically gathered from domain stakeholders, analysts play a key role in refining them into structured and complete user stories.

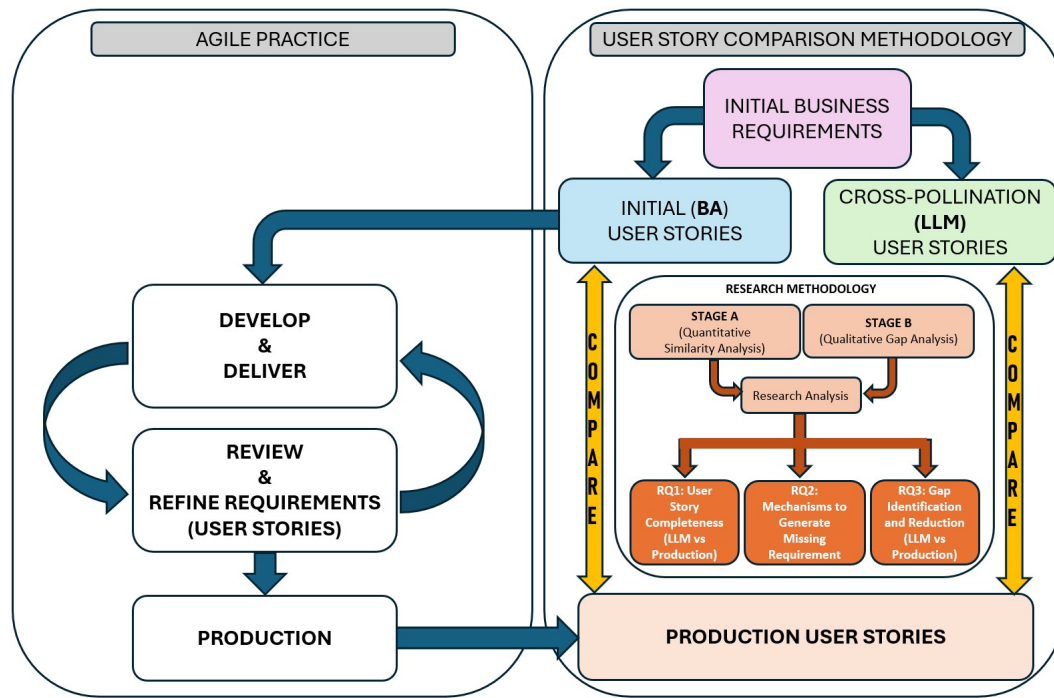
Over the years, RE practices have evolved through various techniques and governance mechanisms aimed at improving requirement quality and completeness [3]. However, software project failures continue to persist [27], with many failures originating from incomplete or inadequate requirements rather than implementation limitations. This highlights the enduring challenge of ensuring that initial requirements sufficiently capture the functionality required for successful system delivery.

The emergence of LLMs introduces new opportunities for supporting RE activities. Unlike traditional approaches that primarily depend on individual analyst expertise, LLMs can leverage broad domain knowledge to identify potential gaps, infer missing capabilities, and provide contextual insights beyond explicitly documented requirements.

Motivated by these challenges and recent advances in LLMs, this study collaborates with industry practitioners to empirically investigate the role of LLM-assisted approaches in identifying requirement gaps and improving completeness. Through evaluation against real-world production-derived artefacts, the research examines both the effectiveness and limitations of LLMs in supporting requirements engineering and reducing downstream software development risks.

Quality Function Deployment (QFD) [2] provides a structured top-down approach for assessing whether engineering specifications satisfy customer intent. However, this study addresses an earlier stage of requirements engineering determining whether initial business requirements are sufficiently complete before specification development. To address this challenge, Cross-Pollination adopts a complementary bottom-up approach that combines structured gap analysis with LLM-assisted contextual reasoning to enrich incomplete requirements.

To address this challenge, this study introduces Cross-Pollination, a domain-agnostic methodology that transforms incomplete natural language business requirements into production-grade functional user stories. Complementary to the top-down perspective of QFD, Cross-Pollination adopts a bottom-up approach that combines structured gap analysis with LLM-assisted contextual reasoning, leveraging both BRD context and domain knowledge embedded within pre-trained LLMs to identify missing capabilities and enrich requirements. The resulting user stories provide a more comprehensive foundation for downstream software development.



■ **Figure 1** User story comparison and comparative analysis methodology.

To evaluate the effectiveness of the proposed methodology, this study benchmarks LLM-generated user stories against realistic production artefacts. It investigates the ability of LLMs to bridge the gap between business analyst (BA) authored requirements and production-ready specifications by identifying functional requirements that may remain implicit or overlooked during human-driven elicitation.

We adopt a three-source comparative evaluation framework (see Figure 1), where analyst-authored and LLM-generated user stories are evaluated against production-derived user stories as the reference baseline. The same Business Requirements Document (BRD) provided to analysts during elicitation is supplied to the LLM, enabling a controlled comparison of human-driven and LLM-assisted generation. This design facilitates the assessment of requirement completeness, coverage, and alignment with functionality implemented in real-world production systems.

These observations motivated the development of a structured methodology in which the LLM first establishes domain understanding, identifies gaps across defined System Capability Dimensions, calibrates expected story coverage, and generates additional requirements to improve completeness. The methodology involves two stages: (1) Cross-Pollination and (2) Comparative Evaluation of User Stories.

- **Cross-Pollination Methodology:** A domain-agnostic, ten-stage framework that transforms incomplete business requirements into production-aligned functional user stories using structured gap analysis and LLM-assisted reasoning.
- **Comparative User Story Evaluation:** A production-derived benchmarking approach that quantitatively evaluates analyst-authored and LLM-generated user stories for completeness and requirement gaps.

To carry out an empirical investigation, we subsequently employed this methodology on three real-world software systems from distinct domains: an E-Commerce application, a Library Management Solution and a Cosmetics and Wellness Education Portal. To evaluate

the outcomes of these approaches, the study formulates a set of research questions. These questions guide the analysis by examining how effectively LLM-assisted techniques identify requirement gaps, improve completeness, and support decision-making during requirements elicitation. The study is structured around research questions:

- **RQ1:** To what extent are LLM-generated user stories complete when benchmarked against production user stories?
- **RQ2:** What qualitative mechanisms enable LLMs to generate comprehensive user stories from Business Requirements Documents (BRD)?
- **RQ3:** What gaps exist between analyst-authored and production user stories, and to what extent can LLMs reduce these gaps?

The findings demonstrate that requirements incompleteness follows structured patterns across domains and that LLM-assisted elicitation complements analyst expertise by improving requirement coverage while requiring human validation to maintain precision.

2 Background and Related Work

Requirements engineering as a continuous and high-risk activity

Requirements engineering (RE), although often treated as a discrete initial phase in the software development lifecycle, is inherently a continuous and evolving stage. In practice, there is a strong desire to stabilise and freeze the requirements early to ensure consistency across subsequent stages of development; however, this objective is rarely achieved. Requirements evolve due to changing stakeholder needs, emerging business priorities and dynamic operational environments, extending RE beyond its initial phase to persist throughout the lifecycle and even post-deployment. Even in stable production systems, new features, enhancements, updates and modifications continue to emerge, reflecting the ongoing evolution of requirements. This continuous nature of RE introduces additional complexity, necessitating effective mechanisms to manage change while maintaining consistency and traceability.

The consequences of requirements deficiencies are considerable and are well documented. Synthesis across multiple empirical programmes indicates that approximately 31% of corporate software projects are cancelled before completion, and a further 53% exceed their original cost estimates by close to 200%. Broader failure rate estimates range from 67% to 85% across different organisational contexts, with 5%–15% of projects being abandoned either before or shortly after delivery due to severe inadequacies. Additional studies report that approximately 20% of projects fail outright, with another 46% experiencing high cost and schedule overruns [50]. For instance, Arianespace suffered from several specifications and design errors, and the failure has been attributed to specification and design errors, including issues related to requirements and requirement changes that propagated into system design. Similarly, a Sydney Water Board project incurred high costs and schedule overruns due to poorly defined requirements and inadequate planning, with rework costs exceeding the initial budget. These findings are further supported by industry analyses, including reports from the Standish Group [50].

More recent evidence from the CHAOS reports [14] indicates that these challenges persist, and the latest trends are not promising either, as only 31% of projects are classified as successful, 50% are challenged, and 19% fail, with large-scale projects exhibiting success rates below 10%. Historical case studies further reinforce these observations.

Fernández et al., [17] in one of the largest global empirical studies in RE have delivered a key insight that many of the most severe issues in software projects originate during early RE activities, particularly due to incomplete, hidden, or poorly elicited requirements. These

problems are not isolated; rather, they propagate throughout the software lifecycle, leading to rework, delays, increased costs, and, in many cases, project failure. The study further emphasises that deficiencies in communication and stakeholder knowledge during elicitation contribute to these issues, reinforcing the complexity of accurately capturing organisational intent. In this context, RE emerges as a foundational and high-risk activity, where early inaccuracies have long-term consequences that are difficult and costly to correct. Therefore, the effectiveness of requirements engineering, especially elicitation and validation, directly influences the alignment between stakeholder needs and system behaviour, making it a critical aspect of software engineering.

Evolution from Traditional Practices to Advanced Approaches

Requirements elicitation has evolved from manual and expert-dependent techniques to AI-driven approaches capable of processing complex, large-scale artefacts. Early methods surveyed by Zowghi et al. [57] remain foundational: interviews [1, 24] enable rapid information gathering but depend heavily on interviewer skill [18], while structured variants such as Volere [41] offer consistency at the cost of flexibility. Complementary techniques including task analysis [13, 40], introspection [18], repertory grids [12], CRC cards [8], and laddering [23] support structured domain understanding. Collaborative brainstorming requirements workshops [19], and focus groups [28] facilitate group-driven exploration, while ethnographic methods [6], protocol analysis [18, 35], and apprenticing [42, 9] provide deeper contextual insight [34]. Goal-oriented and scenario-based approaches like CREWS, Inquiry Cycle [49] decompose objectives into refined requirements [43, 38, 21], while viewpoint-oriented methods [46, 36] capture multi-stakeholder perspectives for more comprehensive specifications.

The increasing complexity of modern systems has driven the adoption of computational techniques to augment these traditional methods [11]. As documented by Popoola et al. [37], NLP has emerged as a key enabler, automating the analysis of textual requirements artefacts [56] and addressing the inefficiency and error-proneness of manual approaches. NLP techniques parse unstructured text, extract entities, detect implicit requirements, and identify inconsistencies [54, 7]. ML (Machine Learning) models further extended these capabilities through historical data-driven prediction of requirement evolution, while HCI techniques enhance stakeholder engagement via interactive interfaces and prototyping [45, 15]. Together, these advances underscore the need for integrated models that improve requirement quality and completeness [39].

The House of Quality (HoQ), a core component of Quality Function Deployment (QFD) [2], captures the Voice of the Customer (VoC)[20] and maps customer requirements to engineering characteristics through weighted relationships. This mapping acts as a transfer function for evaluating specification coverage, with approaches such as the Convergence Gap [31] providing measures of completeness.

The emergence of large language models (LLMs) has further expanded the scope of requirements engineering beyond task-specific NLP. Trained on diverse, cross-domain data, LLMs can surface non-obvious patterns and functional expectations, yet remain inherently uncertain and difficult to control [25, 55], producing inconsistent outputs that raise trust concerns [44, 16]. This has driven focus on prompt engineering (PE), the systematic design of inputs to guide LLM behaviour [33] encompassing techniques such as few-shot prompting [10], chain-of-thought reasoning [51], retrieval-augmented generation (RAG) [29], constraint injection, and multi-role prompting. Despite their promise, these approaches remain fragmented in RE contexts, often lacking holistic evaluation frameworks [26]. Knowledge-

augmentation methods such as RAG [29], ReAct [52], and Chain-of-Knowledge prompting [30] offer further reliability by grounding outputs in external domain knowledge, improving both relevance and trustworthiness.

3 Research Gaps and Motivation

In practice, initial business requirements are captured at a high level, when stakeholders rarely possess a complete understanding of the required functionality. As development progresses, additional business rules, workflows, validations, and exception scenarios inevitably emerge. Consequently, the challenge is not solely whether engineering specifications satisfy customer intent, but whether the initial business requirements are sufficiently complete to derive a production-ready functional specification.

Recent literature identifies domain understanding, hallucination, and limited industry validation as persistent challenges in applying LLMs to Requirements Engineering. Hemmat et al. [22] highlight difficulties with domain-specific language, hallucinations, complex dependencies, and the lack of actionable guidance for LLM adoption in RE, while Arora et al. [4] identifies domain understanding as a critical factor influencing LLM effectiveness in SWOT analysis. Although LLMs can acquire broad domain knowledge, they may not capture the tacit expertise required for accurate requirements elicitation. Furthermore, Zadenoori et al. [53] report that 76% of LLM4RE studies rely on laboratory evaluations, with only 7% involving field studies, indicating limited industrial validation. Existing industry evaluations also largely depend on manual validation of individual stories against acceptance criteria rather than systematic completeness assessment [48]. The absence of realistic production-derived benchmarks and reproducible domain-agnostic evaluation methods remains a key limitation.

These gaps motivate the need for a methodology that can (i) evaluate requirement completeness against production-grounded references, (ii) support domain-aware yet domain-agnostic requirement discovery, and (iii) quantify the gap between documented requirements and implemented functionality. To address these needs, Cross-Pollination combines structured System Capability Dimension-based gap analysis with LLM assisted contextual reasoning to identify implicit and omitted functional requirements. Unlike QFD, which begins with a structured Voice of the Customer exercise, Cross-Pollination operates on an existing BRD the primary artefact available at the start of software development and serves as a complementary downstream approach for enriching requirements before specification development. The methodology further establishes a production derived evaluation framework to benchmark analyst-authored and LLM generated user stories, enabling objective assessment of completeness beyond subjective validation.

4 Methodology

The proposed methodology comprises two complementary components aligned with the research questions: (i) the Cross-Pollination methodology, which transforms incomplete BRD into production-grade functional user stories through structured gap analysis and LLM-assisted contextual reasoning (RQ2); and (ii) a comparative evaluation framework that benchmarks analyst-authored and LLM-generated user stories against production-derived user stories to quantify completeness and analyse requirement gaps (RQ1 and RQ3).

Preliminary experiments were conducted using a Ticket Booking System to examine LLM behaviour under different prompting strategies. Providing individual requirements produced detailed user stories but offered limited additional value, while providing the complete BRD

without structured guidance resulted in insufficient coverage. Multi-shot prompting improved coverage but introduced example-dependent bias. Therefore, a structured zero-shot approach was adopted to provide a consistent and domain-agnostic evaluation setting.

Hence, the methodology employs zero-shot prompting, whereby the LLM receives only the Cross-Pollination prompt and the BRD as input. This configuration isolates the effectiveness of the proposed methodology while providing a consistent basis for comparison across datasets. Cross-Pollination consists of ten sequential stages, each building upon the outputs of preceding stages. Consequently, isolating the contribution of individual stages through conventional ablation is challenging, as removing or modifying an earlier stage changes the contextual information available to subsequent stages and affects the behaviour of the complete pipeline. Therefore, Cross-Pollination is evaluated as an integrated methodology rather than through isolated component analysis.

To support reproducibility, the complete replication package, including the Cross-Pollination prompt suite, datasets, workflow documentation, and evaluation scripts, is publicly available through the accompanying GitHub repository¹. Practitioners require only a Business Requirements Document (BRD) as input, while researchers reproducing the evaluation additionally require a production-derived user story reference set to compute the reported completeness and comparison metrics.

4.1 Cross-Pollination Methodology

The Cross-Pollination methodology is based on the observation that requirement incompleteness is structural rather than random, with high-level business requirements often under-specifying functional details that emerge during implementation. Our experiments demonstrate that directly prompting an LLM to generate user stories from a BRD frequently produces incomplete coverage and hallucinated requirements due to the absence of structured guidance.

To address this, Cross-Pollination adopts a structured ten-stage, zero-shot methodology that guides the LLM through domain understanding, gap identification, scope calibration, and user story generation. Rather than generating requirements directly, the methodology systematically analyses the BRD and cross-references documented requirements with LLM domain knowledge to identify implicit or omitted functionality. The analysis is organised using System Capability Dimensions comprising functional dimensions (Actors, Workflows, Data, Business Rules, States, Integrations, System Events, and Reporting) and regulatory dimensions (Compliance, Security and Privacy, and Governance). These dimensions provide a consistent framework for extracting documented requirements from the BRD and identifying domain standard capabilities from LLM to support systematic gap analysis.

The methodology incorporates complexity calibration to estimate the expected range of production-ready user stories and maintains end-to-end traceability by classifying each generated story according to its source: BRD-derived, gap-identified, or LLM-inferred. The first five stages focus on domain understanding and requirement discovery, while the remaining stages perform calibration, generation, validation, and output assembly.

The ten phases of the Cross-Pollination methodology are outlined below:

1. **Domain Reference Scan:** Reviews the Business Requirements Document (BRD) to establish the system context, purpose, stakeholders, operating environment, and scope. Using this understanding, the LLM leverages its domain knowledge to identify

¹ GitHub Repository: <https://github.com/VinayKabadi/Cross-Pollination>

- representative production systems within the same domain and synthesises their common capabilities into a consolidated domain reference model. These capabilities are organised according to the System Capability Dimensions providing a structured baseline for subsequent comparison.
2. **Deep BRD Comprehension:** Systematically analyses the BRD by extracting all explicitly specified requirements and system elements across the System Capability Dimensions, establishing the baseline representation of documented requirements.
 3. **Static Gap Analysis:** Compares the extracted BRD requirements against the domain reference model across the ten gap dimensions (see Table 1) to identify missing or under-specified requirements, producing a structured gap inventory for subsequent requirement discovery.
 4. **Dynamic Cluster Analysis:** Groups the extracted requirements and identified gaps into logical functional clusters by analysing semantic relationships, dependencies, and interactions, establishing the contextual links required for Cross-Pollination.
 5. **Cross-Pollination Matrix:** The core phase of the methodology, where BRD elements are systematically cross-referenced with identified gaps derived from the LLM's domain knowledge and functional reference model. This structured cross-pollination uncovers contextual relationships, dependencies, and missing capabilities, generating functional requirements that are not explicitly documented in the BRD.
 6. **BRD Complexity Calibration:** Estimates the expected range of user stories for a production system by evaluating the BRD across eight functional and structural dimensions (see Table 2), producing a calibrated Floor, Ceiling, and overall Complexity Tier. This brings stability to the number of user stories.
 7. **Gap Report:** Documents all identified gaps, supporting evidence, and their traceability to the generated requirements.
 8. **Story Generation:** Generates production-ready functional user stories within the calibrated range by combining BRD-derived, gap-driven, and LLM-inferred requirements while maintaining full traceability. The generation process applies consistency checks to minimise hallucinations, redundancy, and duplicate user stories.
 9. **Production Readiness Checklist:** Validates the generated user stories for completeness, consistency, coverage, and adherence to production standards.
 10. **Output Assembly:** Compiles the final deliverable, including the generated user stories, traceability matrix, gap report, domain reference model, and coverage summary.
- Cross-Pollination methodology employs two kinds of dimensions, i.) Gap analysis dimensions (see Table 1) and ii.) Calibration dimensions (see Table 2).

Gap Analysis Dimensions

Business requirements typically describe systems in terms of intended operation defining actors, workflows, and data while omitting the elements required for reliable and complete production behaviour. The ten gap analysis dimensions in Cross-Pollination capture this boundary as a structured taxonomy of commonly missing requirement categories, including automated roles, exception paths, data integrity controls, configurable rules, lifecycle states, integration failures, validation constraints, notification triggers, background jobs, and data migration processes.

These ten dimensions are applied in Phase 3 of cross-pollination to form a closed, domain-agnostic framework that is used consistently across all business requirements. No dimension is added, removed, or adjusted for specific domains. The focus remains on functional aspects – to ensure that identified gaps can be expressed as testable user stories and compared uniformly across domains. Table 1 presents the complete taxonomy.

■ **Table 1** The Ten Gap Dimensions Used for Requirements Analysis.

Code	Dimension	What It Measures
D01	Actor Completeness	Named actors, all roles and personas explicitly identified
D02	Workflow Completeness	End-to-end processes fully described including exception paths
D03	Data Completeness	Data entities and integrity mechanisms what is captured and how
D04	Business Rules	Configurable policies, thresholds, and constraints explicitly stated
D05	State Machine	Object lifecycle states and all transitions named
D06	Exception Handling	Failure modes, error paths, and fallbacks covered
D07	Validation	Input constraints and format rules specified
D08	Notification Events	All trigger conditions for notifications named
D09	Background Jobs	Scheduled and automated execution mechanisms specified
D10	Onboarding & Offboarding	Registration, activation, suspension, and deactivation flows

Complexity Calibration Dimensions

The calibration model addresses a key challenge in completeness assessment: without a complexity-based reference, it is difficult to determine whether a backlog is complete or simply large or small. The BRD is evaluated across eight weighted dimensions (six functional and two structural), informed by practitioner experience and observed production patterns. The aggregated score produces a midpoint, from which a Floor ($\times 0.78$) and Ceiling ($\times 1.28$) define the acceptable range of user stories, along with a Complexity Tier for cross-project comparison. Higher weights are assigned to dimensions such as actor count, state complexity, and integrations, as these increase interaction paths, lifecycle scenarios, and system dependencies. The bounds are grounded in the observation that approximately 20% of requirements typically emerge during later refinement; the lower bound accommodates under-specification, while the upper bound allows greater tolerance for over-specification, which is more prevalent once minimum coverage is achieved.

4.2 Comparative User Story Evaluation

Evaluating the quality of requirements artefacts presents a fundamental measurement challenge: unlike source code or test suites, user stories have no universally accepted ground truth against which correctness can be objectively assessed. Prior work has relied predominantly on expert inspection, structured walk-through and checklist-based heuristics, all of which are labour-intensive, subjective and difficult to reproduce across domains.

To address this limitation, we deployed a systematic, domain-agnostic evaluation framework that operationalises requirements quality as measurable functional coverage against a production-implemented reference set. The method comprises two sequential analytical stages: Stage A an automated similarity evaluation analysis that produces quantitative coverage metrics, and Stage B a structured gap analysis that diagnoses the nature and

■ **Table 2** Calibration Dimensions and Scaling Formulas.

Code	Calibration Dimension	Formula
D-C01	Actor Count	actors $\times 4$
D-C02	Workflow Density	decision points $\times 1.8$
D-C03	Business Rules	rules $\times 1.5$
D-C04	State Machine Depth	states $\times 2$
D-C05	Integration Count	integrations $\times 3$
D-C06	Notification Surface	events $\times 1.2$
D-C07	Reporting Depth	reports $\times 1.8$
D-C08	Distributed Network	Single / Multi

distribution of coverage gaps. Together the LLM answers both, in Stage A, it answers how much of the production system each candidate backlog predicts, and in Stage B it answers which requirements were missed, why they were missed, and what structural categories they belong to.

To evaluate we gather three data sets for each project: **PROD**, **INITIAL** and **LLM**. PROD serves as the reference set, comprising production-derived user stories reconstructed by industry professionals through multiple approaches, including recreating stories from the live production system, reverse engineering from source code and API routes, and consolidating artefacts from project boards across the entire development lifecycle. Unlike specification documents, which may describe intended rather than realised behaviour, production artefacts provide an unambiguous record of implemented functionality. INITIAL comprises user stories produced by a business analyst from a BRD, while LLM comprises stories generated by a Large Language Model from the same BRD. Comparing both candidate sets against the shared PROD reference isolates the effect of the elicitation method while controlling for input document, domain, and evaluation instrument.

Stage A – Quantitative Similarity Analysis

The matching process is carried out in five steps. Each step prepares the stories, compares them, and evaluates how well they match production.

Step 1: Text Normalisation. All stories are normalised by applying three sequential stop-word layers to the story body text: (i) standard English function words; (ii) universal user story boilerplate (e.g., *as*, *want*, *so*, *that*, *can*); and (iii) domain-specific high-frequency vocabulary. The story title is excluded to avoid naming bias. The remaining tokens form the semantic representation used for matching.

Step 2: TF-IDF Vectorisation. Each normalised story is converted to a weighted term vector using Term Frequency–Inverse Document Frequency (TF-IDF) [47] which down-weights common terms while emphasising discriminative, domain-specific vocabulary. The inverse document frequency for term t is computed as:

$$\text{IDF}(t) = \log \left(\frac{N + 1}{df(t) + 1} \right) + 1$$

where N is the total number of user stories in the corpus and $df(t)$ is the number of stories containing term t . The smoothing terms (+1) prevent division by zero and ensure stable weighting for infrequent terms.

Step 3: Cosine Similarity. Cosine similarity measures the semantic similarity between two TF-IDF vectors and is well suited to short-text comparison because it is invariant to document length [32]. Given two TF-IDF vectors A and B :

$$\text{cosine}(A, B) = \frac{A \cdot B}{\|A\| \cdot \|B\|}$$

where $A \cdot B$ denotes the dot product and $\|A\|$ and $\|B\|$ are the Euclidean norms of the corresponding vectors. Cosine similarity ranges from 0 (no shared semantic content) to 1 (identical vector representations). A supplementary functional-area boost of +0.03 is applied when two stories share the same epic code, rewarding domain alignment without overriding the primary semantic similarity signal. Average Match Score is reported as the mean similarity of all matched pairs.

Step 4: Find the best one-to-one matches. Greedy matching is applied by assigning each PROD user story to the highest-scoring unmatched candidate, provided the similarity score exceeds a configurable threshold of 0.10. The low threshold was selected heuristically because independently authored requirements often use different terminology to describe the same functionality [5], and was reviewed by industry practitioners to provide a reasonable balance between recall and valid semantic matching. Its impact can be further evaluated through sensitivity analysis in future work.

Step 5: Metric Calculation. Matching performance is evaluated using Precision, Recall, and F1-score. A True Positive (TP) represents a correctly matched user story, a False Positive (FP) is a generated story with no corresponding production match, and a False Negative (FN) is a production story with no corresponding generated match. Precision measures the proportion of generated stories that are correct, Recall measures the proportion of production stories identified, and the F1-score provides their harmonic of precision and recall. True Negatives (TN) are excluded, as their inclusion would artificially inflate performance in this retrieval-style evaluation.

$$\text{In-Scope (Precision)} = \frac{TP}{TP + FP}$$

$$\text{Coverage (Recall)} = \frac{TP}{TP + FN}$$

$$F1 = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

Stage B – Qualitative Gap analysis

This stage serves as the interpretive complement to Stage A’s quantitative evaluation. While Stage A measures *how much* of the production system is captured by each source, Stage B focuses on understanding *which* requirements were missed, *why* they were missed, and *what structural categories* are systematically absent. It operates on the unmatched story sets produced in Step 4 of Stage A and analyses them through four independent steps, each addressing a progressively deeper diagnostic question about the nature of the gaps. These steps collectively provide a structured explanation of coverage deficiencies and are summarised below.

Step 1: Blind Spot Analysis. This step identifies where coverage gaps exist by examining production (PROD) stories that were not matched by candidate sets. These unmatched stories are grouped by functional area (epics) to distinguish between scattered gaps – where coverage exists but is incomplete – and concentrated gaps, where entire functional areas are missing. This helps separate minor omissions from fundamental misunderstandings of system scope.

Step 2: FP Classification. This step reassesses unmatched candidate stories (False Positives) to determine whether they are truly incorrect or simply outside current production scope. Stories are classified into three categories: Aspirational (valid but not yet implemented), Over-generated (irrelevant or unnecessary), and Future scope (plausible extensions). This classification refines the interpretation of precision by recognising that not all unmatched stories are noise.

Step 3: BRD Cross-reference. In this step, each section undergoes a review to identify missing or under-represented requirements. The analysis highlights role gaps (missing actors), requirement gaps (unstated functionality), and code-only features (implemented behaviour not documented in the BRD), providing a deeper understanding of specification completeness.

Step 4: Gold Standard Synthesis. In the final step, findings from all previous analyses are combined to construct a comprehensive “gold standard” backlog. This backlog includes production-derived user stories as a baseline reference set, supplemented with missing requirements identified in earlier steps and enriched descriptions where applicable. Each story is tagged based on its origin (e.g., PROD, BRD gap, inferred), ensuring full traceability and enabling a complete and validated representation of system requirements.

5 Experiments and Results

To evaluate the proposed methodology², three industry projects were selected in which the participating practitioners, who are also the authors, were directly involved: (i) an E-Commerce platform, (ii) a Library Management System (LMS), and (iii) a wellness and cosmetic beauty application developed by Trawelli Pty Ltd. For each project, three datasets were established: INITIAL, comprising analyst-authored user stories derived from the original Business Requirements Document (BRD); PROD, representing the final implemented functionality; and LLM, comprising user stories generated using the proposed methodology.

While the INITIAL dataset was obtained from project artefacts, the PROD dataset was reconstructed because no single consolidated source existed and requirements had evolved throughout development. Reconstruction was performed by industry practitioners using backlog artefacts, product documentation, domain knowledge, and, where necessary, reverse engineering from the implemented systems. The reconstructed user stories were subsequently validated to ensure consistency of functional intent.

The LLM dataset was generated using Claude Sonnet 4.6 (web) in a zero-shot configuration, relying solely on the initial BRD. Consequently, both the analyst-authored and LLM-generated user stories were derived from the same source requirements, enabling a controlled comparison. Overall, two-thirds of the evaluation dataset originated from industrial artefacts, while one-third was generated under controlled LLM conditions.

² GitHub Repository: <https://github.com/VinayKabadi/Cross-Pollination>

The study evaluates the completeness of business analyst (BA) authored and LLM-generated user stories by benchmarking them against the PROD dataset, which serves as the ground truth. The INITIAL user stories represent the business analyst's initial interpretation of the Business Requirements Document (BRD), while the LLM outputs provide an independent, model-driven perspective. Experiments are conducted across all three projects, evaluating user stories against the ten System Capability Dimensions to identify requirement gaps, followed by a comparative assessment of BA (Business Analyst) and LLM performance relative to the production baseline.

RQ 1: To what extent are LLM-generated user stories complete when benchmarked against production user stories?

■ **Table 3** E-Commerce: Comparison of BA and LLM user stories against PROD.

Metric	INITIAL (BA) vs PROD	LLM vs PROD	LEADER
<i>User Stories</i>	47 vs 122	258 vs 122	–
In-Scope (Precision)	97.9%	46.5%	BA
Coverage (Recall)	37.7%	98.4%	LLM
F1	54.4%	63.2%	LLM

■ **Table 4** LMS (Library Mgmt System): Comparison of BA and LLM user stories against PROD.

Metric	INITIAL (BA) vs PROD	LLM vs PROD	LEADER
<i>User Stories</i>	47 vs 110	224 vs 110	–
In-Scope (Precision)	93.6%	46.4%	BA
Coverage (Recall)	40.0%	94.5%	LLM
F1	56.1%	62.3%	LLM

■ **Table 5** Wellness app (Trawelli): Comparison of BA and LLM user stories against PROD.

Metric	INITIAL (BA) vs PROD	LLM vs PROD	LEADER
<i>User Stories</i>	160 vs 220	222 vs 220	–
In-Scope (Precision)	100.0%	78.8%	BA
Coverage (Recall)	72.7%	79.5%	LLM
F1	84.2%	79.2%	BA

Across all three projects, BA-generated stories maintain very high rate for being in scope with a high precision range of 93.6%–100%, indicating that most of their stories are directly relevant to production. In contrast, LLM-generated user stories achieve lower precision scores, ranging from 46.4% to 78.8%, reflecting the generation of additional stories that do not map directly to implemented functionality. This suggests that while LLMs significantly expand coverage, they also introduce a higher proportion of exploratory requirements which, might be relevant, fall outside the scope of production and may be considered as noise.

But LLM-generated user stories consistently achieve significantly higher coverage (recall) compared to analyst-authored (BA) stories, indicating a stronger ability to capture the full scope of production functionality. In the E-Commerce (refer to Table 3), LLM recall reaches 98.4% compared to 37.7% for BA, while for LMS (refer to Table 4), it achieves 94.5% versus 40.0%. A similar pattern is observed for the Wellness App (Trawelli) (refer to Table 5), where LLM recall (79.5%) exceeds BA recall (72.7%). This demonstrates that LLMs are highly effective in identifying missing requirements and approximating production-level completeness.

82:14 Can LLMs Identify Missing Elements in Requirements Elicitation?

When considering the F1 score, which captures the trade-off between precision and recall, LLMs outperform BA in two of the three domains (E-Commerce: 63.2% vs 54.4%, LMS: 62.3% vs 56.1%), indicating a better overall balance between completeness and relevance. In the dataset from Trawelli, BA achieves a slightly higher F1 score (84.2% vs 79.2%), primarily due to near-perfect precision, though at the cost of lower coverage. Trawelli results differ because the development scope was tightly controlled, with no changes accepted into the current release. Additional or emerging requirements were deferred to next development stages, resulting in BA outputs that closely align with the implemented scope (precision of 100%) but exclude future-stage requirements, leading to lower overall coverage (72.7%).

Overall, the results indicate that LLM-generated user stories provide substantially higher completeness relative to production systems, making them effective for uncovering missing requirements. However, this completeness is achieved by generating a broader set of stories, requiring further filtering or validation to ensure precision or to identify the noise.

RQ 2: What qualitative mechanisms enable LLMs to generate comprehensive user stories from Business Requirements Documents (BRD)

The results presented in Table 6 reveal that LLM-generated stories identify substantially more requirement gaps than analyst-authored stories across all ten dimensions and all three evaluation domains. In aggregate, the BA identified 40, 34 and 38 gaps across E-Commerce, LMS and Trawelli respectively, while the LLM contributed an additional (+) 84, 79 and 78 gaps beyond those identified by the BA, an aggregate gap discovery ratio of approximately 3:1 in favour of LLM generation across all domains. These results confirm the central hypothesis of this research: LLMs do not simply generate more requirements, they recover a class of requirements that were not identified during elicitation from the same BRD.

■ **Table 6** Comparison of gaps identified by BA and LLM across three domains.

Dimension	E-COMMERCE		LMS		TRAWELLI	
	BA	LLM	BA	LLM	BA	LLM
D01: Actor Completeness	5	+2	4	+3	4	+1
D02: Workflow Completeness	14	+9	12	+7	14	+8
D03: Data Completeness	9	+4	7	+5	8	+5
D04: Business Rules	3	+13	0	+9	2	+9
D05: State Machine	5	+21	5	+17	3	+19
D06: Exception Handling	1	+6	0	+7	0	+7
D07: Validation	0	+8	0	+6	0	+5
D08: Notification Events	3	+8	6	+7	4	+6
D09: Background Jobs	0	+9	0	+10	2	+11
D10: Onboarding/Offboarding	0	+4	0	+8	1	+7
TOTALS	40	+84	34	+79	38	+78

The results indicate that business analysts primarily capture requirements explicitly documented in the BRD, particularly Actors (D01), Workflows (D02), and Data (D03), which are typically articulated by stakeholders during elicitation. In contrast, the largest improvements achieved by Cross-Pollination occur in dimensions that are often implicit within Business Requirements, including business rules, state transitions, validation constraints, exception handling, background processes, and onboarding. These aspects are rarely documented as standalone requirements and instead emerge from the interactions between actors, workflows, and business context. The findings therefore suggest that LLM-assisted reasoning complements human elicitation by systematically inferring operational behaviours and constraints required for production-ready systems rather than simply generating additional user stories.

The results also suggest that Actors (D01) and Workflows (D02) function as foundational dimensions from which many other requirements emerge. Business rules, validations, exception paths, and background processes are typically associated with specific actors performing defined workflows. Consequently, omissions in these dimensions can propagate across multiple dependent capabilities, whereas more complete actor and workflow definitions provide a stronger foundation for identifying implicit requirements. Although this cascading relationship requires further empirical validation, it offers a plausible explanation for the consistently higher completeness achieved by Cross-Pollination.

Across all three industry projects, despite differences in domain and system complexity, the same structural pattern was observed. These findings suggest that business analysts and LLMs should be viewed as complementary rather than competing approaches. Analysts contribute stakeholder understanding and domain expertise, while LLMs systematically expand requirement coverage by identifying implicit functional requirements. Together, they suggest a more comprehensive foundation for production-ready functional specifications than either approach in isolation.

RQ3: What gaps exist between analyst-authored and production user stories, and to what extent can LLMs reduce these gaps?

The results reveal a consistent gap between analyst-authored (INITIAL) and production (PROD) user stories, primarily arising from missing requirements in structurally under-specified areas. While BA outputs provide strong coverage of explicit elements such as actors, primary workflows, and data entities, they consistently under-represent business rules, state transitions, validation, exception handling, background jobs, and onboarding processes, all of which are essential for production-ready systems.

LLM-generated user stories substantially reduce these gaps by identifying implicit functional requirements across all System Capability Dimensions. As demonstrated in RQ1, this leads to consistently higher recall, particularly in dimensions that are sparsely represented in BA outputs, indicating that LLMs are effective at uncovering missing functional capabilities beyond those explicitly documented in the BRD. However, this improvement is accompanied by lower precision as some generated requirements represent plausible but non-implemented functionality. Such user stories should be treated as candidates requiring validation by business analysts and stakeholders before incorporation into the product backlog.

Overall, the findings indicate that LLMs substantially reduce the gap between analyst-authored and production user stories by improving functional completeness, while human review remains essential to ensure alignment with stakeholder intent and production scope.

Summary of Key Findings and Results

RQ1 – Completeness of LLM-generated user stories. LLM-generated user stories achieve higher completeness than analyst-authored stories, with recall ranging from 79.5%–98.4% compared with 37.7%–72.7% for BA. However, this increased coverage is accompanied by lower precision (46.4%–78.8% versus 93.6%–100%), due to additional plausible but non-production requirements.

RQ2 – Mechanisms enabling comprehensive user story generation. Cross-Pollination improves requirement discovery by combining structured gap analysis, contextual reasoning, and LLM domain knowledge to identify implicit requirements, particularly business rules, validations, exceptions, and operational workflows that are frequently under-represented in BA-authored user stories.

RQ3 – Gap reduction between analyst-authored and production user stories. Cross-Pollination reduces the gap between analyst-authored and production user stories by recovering systematically omitted functional requirements. However, human validation remains necessary to ensure alignment with stakeholder intent and production scope.

Overall Finding. LLMs complement rather than replace analysts by improving requirement completeness, while analysts provide the judgement required to maintain precision and relevance.

6 Threats to Validity

The study relied exclusively on the Claude Sonnet 4.6 web interface, selected based on the industry practitioners' experience at the time of the research, which may limit generalisability. For API-based replication, the closest equivalent configuration, based on Anthropic guidance, is a temperature of 0.2, a maximum output of 16,000 tokens, and a 200,000-token context window. The evaluation employed zero-shot prompting; however, evolving model versions may affect exact reproducibility. Furthermore, LLM non-determinism and hallucination may influence the generation of plausible but non-aligned requirements. Such behaviour may contribute to reduced precision and functionalities outside the intended scope.

The methodology assumes that the BRD reasonably reflects stakeholder intent. Similar to QFD, where outcomes depend on the quality of the Voice of the Customer (VoC), Cross-Pollination relies on the available requirements context while using domain knowledge to complement missing information. Ambiguous inputs may result in gaps being identified against an incorrect baseline or generate requirements beyond the intended scope. Furthermore, although designed as domain-agnostic, its applicability may vary across domains with different documentation practices, terminology, and regulatory contexts, and the curated datasets may not capture the full diversity of real-world requirements.

The evaluation involves human judgement in validating requirements, reviewing outputs, and constructing ground truth, which may introduce reviewer bias. Although structured validation procedures and source classification help mitigate this risk, expert review remains necessary before incorporating generated requirements into production backlogs.

Finally, the similarity threshold used for automated user story matching was selected heuristically and reviewed by industry practitioners rather than empirically optimised. Different thresholds may influence precision, recall, and overall evaluation outcomes; therefore, sensitivity analysis is required to further assess its impact.

7 Future Work

An open question is whether LLM outputs reflect genuine reasoning or statistical pattern matching, particularly in domains that are well represented in public training data. Evaluating the methodology across niche and proprietary domains with limited exposure would help assess its robustness and generalisation capability.

Further investigation should evaluate the Cross-Pollination methodology using multiple state-of-the-art LLMs (e.g., GPT, Gemini, Llama, and Mistral) and alternative prompting strategies (e.g., one-shot and few-shot) to assess its sensitivity to different models and prompting configurations. Exploring domain-specific fine-tuning, human-in-the-loop refinement, and structured requirement representations could further improve the robustness, accuracy, and broader applicability of the approach.

Additional evaluation should examine the sensitivity of user story matching thresholds by analysing their impact on precision, recall, and overall matching performance across diverse datasets. Standardised evaluation frameworks could further enhance reproducibility and facilitate comparison across different approaches.

A further research direction is to investigate the completeness–precision trade-off in practical settings by measuring practitioner validation effort, acceptance rates of LLM-generated requirements, and effective strategies for integrating LLM-assisted requirement generation into existing requirements engineering workflows.

8 Conclusion

This study aimed to improve the identification of incompleteness in Business Requirements Document (BRD) and support the derivation of more complete functional specifications through the proposed Cross-Pollination methodology. By benchmarking analyst-authored and LLM-generated user stories against production-derived ground truth, the study presents a reproducible, domain-agnostic framework for generating and evaluating user stories, enabling measurable assessment of requirement coverage.

While Cross-Pollination improves requirement completeness, generated requirements should be treated as candidate requirements rather than confirmed stakeholder needs. The methodology augments, rather than replaces, business analysts by combining human expertise with LLM-assisted discovery. Organisations can apply Cross-Pollination after establishing an initial BRD, where analysts validate, prioritise, and integrate relevant requirements into existing workflows.

Across all evaluated domains, analyst-authored stories demonstrate high precision but limited recall, capturing explicitly documented requirements while missing functionality implemented in production systems. In contrast, LLM-generated stories achieve higher recall by identifying implicit and omitted requirements, although with reduced precision due to plausible but non-implemented or out-of-scope scenarios. This completeness–precision trade-off positions LLMs as requirement discovery assistants rather than autonomous elicitation tools, requiring additional validation effort.

Overall, the findings demonstrate the value of a human–LLM collaborative approach to requirements engineering, where analysts provide contextual judgement and validation while LLMs support broader requirement exploration. Cross-Pollination operationalises this collaboration through structured gap analysis and traceable evaluation, supporting more complete and production-aligned functional specifications.

Generative AI Declaration

Generative AI tools were used to improve the readability and language of this manuscript. All AI-generated content was reviewed, verified, and edited by the authors, who take full responsibility for the final published work.

References

- 1 Ritu Agarwal and Mohan R Tanniru. Knowledge acquisition using structured interviewing: an empirical investigation. *Journal of Management Information Systems*, 7(1):123–140, 1990. doi:10.1080/07421222.1990.11517884.
- 2 Yoji Akao. *Quality function deployment: integrating customer requirements into product design*. Productivity press, 1990.

- 3 Bogdan-Alexandru Andrei, Andrei-Cosmin Casu-Pop, Sorin-Catalin Gheorghe, and Costin-Anton Boiangiu. A study on using waterfall and agile methods in software project management. *Journal of Information Systems & Operations Management*, pages 125–135, 2019.
- 4 Chetan Arora, John Grundy, and Mohamed Abdelrazek. Advancing requirements engineering through generative ai: Assessing the role of llms. In *Generative AI for Effective Software Development*, pages 129–148. Springer, 2024.
- 5 Chetan Arora, Mehrdad Sabetzadeh, Lionel Briand, and Frank Zimmer. Automated checking of conformance to requirements templates using natural language processing. *IEEE transactions on Software Engineering*, 41(10):944–968, 2015. doi:10.1109/TSE.2015.2428709.
- 6 Linden J Ball and Thomas C Ormerod. Putting ethnography to work: the case for a cognitive ethnography of design. *International Journal of Human-Computer Studies*, 53(1):147–168, 2000. doi:10.1006/IJHC.2000.0372.
- 7 Dipali Baviskar, Swati Ahirrao, Vidyasagar Potdar, and Ketan Kotecha. Efficient automated processing of the unstructured documents using artificial intelligence: A systematic literature review and future directions. *Ieee Access*, 9:72894–72936, 2021. doi:10.1109/ACCESS.2021.3072900.
- 8 Kent Beck and Ward Cunningham. A laboratory for teaching object oriented thinking. *ACM Sigplan Notices*, 24(10):1–6, 1989. doi:10.1145/74877.74879.
- 9 Hugh R Beyer and Karen Holtzblatt. Apprenticing with the customer. *Communications of the ACM*, 38(5):45–52, 1995. doi:10.1145/203356.203365.
- 10 Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.
- 11 Faiza Allah Bukhsh, Zaharah Allah Bukhsh, and Maya Daneva. A systematic literature review on requirement prioritization techniques and their empirical evaluation. *Computer Standards & Interfaces*, 69:103389, 2020. doi:10.1016/J.CSI.2019.103389.
- 12 Trevor Butt. *George Kelly: The psychology of personal constructs*. Bloomsbury Publishing, 2008.
- 13 Pär Carlshamre and Joachim Karlsson. A usability-oriented approach to requirements engineering. In *Proceedings of the Second International Conference on Requirements Engineering*, pages 145–152. IEEE, 1996. doi:10.1109/ICRE.1996.491439.
- 14 OpenCommons contributors. Chaos report on it project outcomes, 2025. URL: https://opencommons.org/CHAOS_Report_on_IT_Project_Outcomes.
- 15 Chinedu Alex Ezeigweneme, Aniekan Akpan Umoh, Valentine Ikenna Ilojiana, and Abimbola Oluwatoyin Adegbite. Telecommunications energy efficiency: optimizing network infrastructure for sustainability. *Computer Science & IT Research Journal*, 5(1):26–40, 2024.
- 16 Oluwole Fagbohun, Rachel M Harrison, and Anton Dereventsov. An empirical categorization of prompting techniques for large language models: a practitioner’s guide. *arXiv preprint*, 2024. doi:10.48550/arXiv.2402.14837.
- 17 D Méndez Fernández, Stefan Wagner, Marcos Kalinowski, Michael Felderer, Priscilla Mafra, Antonio Vetrò, Tayana Conte, M-T Christiansson, Des Greer, Casper Lassenius, et al. Naming the pain in requirements engineering: Contemporary problems, causes, and effects in practice. *Empirical software engineering*, 22(5):2298–2338, 2017. doi:10.1007/S10664-016-9451-7.
- 18 Joseph A Goguen and Charlotte Linde. Techniques for requirements elicitation. In *[1993] Proceedings of the IEEE International Symposium on Requirements Engineering*, pages 152–164. IEEE, 1993. doi:10.1109/ISRE.1993.324822.
- 19 Ellen Gottesdiener. *Requirements by collaboration: workshops for defining needs*. Addison-Wesley Professional, 2002.
- 20 Abbie Griffin and John R Hauser. The voice of the customer. *Marketing science*, 12(1):1–27, 1993.

- 21 Peter Haumer, Klaus Pohl, and Klaus Weidenhaupt. Requirements elicitation and validation with real world scenes. *IEEE transactions on Software Engineering*, 24(12):1036–1054, 2002.
- 22 Arshia Hemmat, Mohammadreza Sharbaf, Shekoufeh Kolahdouz-Rahimi, Kevin Lano, and Sobhan Y Tehrani. Research directions for using llm in software requirement engineering: A systematic review. *Frontiers in Computer Science*, 7:1519437, 2025.
- 23 David N Hinkle. The change of personal constructs from the viewpoint of a theory of implications. *unpublished doctoral dissertation, Ohio State University*, 1965.
- 24 Karen Holtzblatt and Hugh R Beyer. Requirements gathering: the human factor. *Communications of the ACM*, 38(5):31–32, 1995.
- 25 Haowen Hou, Fei Ma, Binwen Bai, Xinxin Zhu, and Fei Yu. Enhancing and accelerating large language models via instruction-aware contextual compression. *arXiv preprint*, 2024. [arXiv:2408.15491](#).
- 26 Kaicheng Huang, Fanyu Wang, Yutan Huang, and Chetan Arora. Prompt engineering for requirements engineering: A literature review and roadmap. In *2025 IEEE 33rd International Requirements Engineering Conference Workshops (REW)*, pages 548–557. IEEE, 2025. doi: 10.1109/REW66121.2025.00081.
- 27 Azham Hussain, Emmanuel OC Mkpojiogu, and Fazillah Mohmad Kamal. The role of requirements in the success or failure of software projects. *International Review of Management and marketing*, 6(7):306–311, 2016.
- 28 Richard A Krueger. *Focus groups: A practical guide for applied research*. Sage publications, 2014.
- 29 Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, 33:9459–9474, 2020.
- 30 Xingxuan Li, Ruochen Zhao, Yew Ken Chia, Bosheng Ding, Shafiq Joty, Soujanya Poria, and Lidong Bing. Chain-of-knowledge: Grounding large language models via dynamic knowledge adapting over heterogeneous sources. *arXiv preprint*, 2023. [arXiv:2305.13269](#).
- 31 Anthony Lowe and Keith Ridgway. Uk user’s guide to quality function deployment. *Engineering Management Journal*, 10(3):147–155, 2000.
- 32 Christopher D Manning. *Introduction to information retrieval*. Syngress Publishing,, 2008.
- 33 Ggaliwango Marvin, Nakayiza Hellen, Daudi Jjingo, and Joyce Nakatumba-Nabende. Prompt engineering in large language models. In *International conference on data intelligence and cognitive informatics*, pages 387–402. Springer, 2023.
- 34 Karen L McGraw and Karen Harbison-Briggs. *Knowledge acquisition: Principles and guidelines*. Prentice-Hall, Inc., 1989.
- 35 Janni Nielsen, Torkil Clemmensen, and Carsten Yssing. Getting access to what goes on in people’s heads? reflections on the think-aloud technique. In *Proceedings of the second Nordic conference on Human-computer interaction*, pages 101–110, 2002. doi:10.1145/572020.572033.
- 36 Bashar Nuseibeh, Anthony Finkelstein, and Jeff Kramer. Method engineering for multi-perspective software development. *Information and software technology*, 38(4):267–274, 1996. doi:10.1016/0950-5849(95)01054-8.
- 37 Oladapo Adeboye Popoola, Henry Ejiga Adama, Chukwuekem David Okeke, and Abiodun Emmanuel Akinoso. Advancements and innovations in requirements elicitation: Developing a comprehensive conceptual model. *World Journal of Advanced Research and Reviews*, 22(01):1209–1220, 2024.
- 38 Colin Potts, Kenji Takahashi, and Annie I. Anton. Inquiry-based requirements analysis. *IEEE software*, 11(2):21–32, 1994. doi:10.1109/52.268952.
- 39 Aqsa Rasheed, Bushra Zafar, Tehmina Shehryar, Naila Aiman Aslam, Muhammad Sajid, Nouman Ali, Saadat Hanif Dar, and Samina Khalid. Requirement engineering challenges in agile software development. *Mathematical Problems in Engineering*, 2021(1):6696695, 2021.

- 40 Juliet Richardson, Thomas C Ormerod, and Andrew Shepherd. The role of task analysis in capturing requirements for interface design. *Interacting with computers*, 9(4):367–384, 1998. doi:10.1016/S0953-5438(97)00036-2.
- 41 James Robertson and Suzanne Robertson. Volere. *Requirements Specification Templates*, 2000.
- 42 Suzanne Robertson and James Robertson. *Mastering the requirements process: Getting requirements right*. Addison-wesley, 2012.
- 43 Colette Rolland, Carine Souveyet, and Camille Ben Achour. Guiding goal modeling using scenarios. *IEEE transactions on software engineering*, 24(12):1055–1071, 1998. doi:10.1109/32.738339.
- 44 Pranab Sahoo, Ayush Kumar Singh, Sriparna Saha, Vinija Jain, Samrat Mondal, and Aman Chadha. A systematic survey of prompt engineering in large language models: Techniques and applications. *arXiv preprint*, 1, 2024. doi:10.48550/arXiv.2402.07927.
- 45 Yerik Afrianto Singgalen. Coastal and marine tourism monitoring system design using rapid application development (rad). *J. Inf. Syst. Res*, 5(2):468–479, 2024.
- 46 Ian Sommerville, Peter Sawyer, and Stephen Viller. Viewpoints for requirements elicitation: a practical approach. In *Proceedings of IEEE International Symposium on Requirements Engineering: RE'98*, pages 74–81. IEEE, 1998. doi:10.1109/ICRE.1998.667811.
- 47 Karen Sparck Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of documentation*, 28(1):11–21, 1972.
- 48 Tjerk Spijkman, Bente Molenkamp, Steffen Beudeker, Sietse Overbeek, and Fabiano Dalpiaz. Llm-assisted requirements engineering in agile mdd: Industry insights and validation. In *2025 IEEE 33rd International Requirements Engineering Conference (RE)*, pages 366–377. IEEE, 2025. doi:10.1109/RE63999.2025.00043.
- 49 Alistair G Sutcliffe, Neil AM Maiden, Shailey Minocha, and Darrel Manuel. Supporting scenario-based requirements engineering. *IEEE Transactions on software engineering*, 24(12):1072–1088, 1998. doi:10.1109/32.738340.
- 50 June Verner, Jennifer Sampson, and Narciso Cerpa. What factors lead to software project failure? In *2008 Second International Conference on Research Challenges in Information Science*, pages 71–80, 2008. doi:10.1109/RCIS.2008.4632095.
- 51 Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.
- 52 Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *The eleventh international conference on learning representations*, 2022.
- 53 Mohammad Amin Zadenoori, Jacek Dąbrowski, Waad Alhoshan, Liping Zhao, and Alessio Ferrari. Large language models (llms) for requirements engineering (re): A systematic literature review. *arXiv preprint*, 2025. doi:10.48550/arXiv.2509.11446.
- 54 Aya Zaki-Ismail, Mohamed Osama, Mohamed Abdelrazek, John Grundy, and Amani Ibrahim. Rcm-extractor: an automated nlp-based approach for extracting a semi formal representation model from natural language requirements. *Automated Software Engineering*, 29(1):10, 2022. doi:10.1007/S10515-021-00312-Y.
- 55 XiangRui Zhang, XueJie Du, HaoYu Chen, Yongzhong He, Wenjia Niu, and Qiang Li. Automatically generating rules of malicious software packages via large language model. In *2025 55th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 734–747. IEEE, 2025. doi:10.1109/DSN64029.2025.00072.
- 56 Liping Zhao, Waad Alhoshan, Alessio Ferrari, Keletso J Letsholo, Muideen A Ajagbe, Erol-Valeriu Chioasca, and Riza T Batista-Navarro. Natural language processing for requirements engineering: A systematic mapping study. *ACM Computing Surveys (CSUR)*, 54(3):1–41, 2021. doi:10.1145/3444689.
- 57 Didar Zowghi and Chad Coulin. Requirements elicitation: A survey of techniques, approaches, and tools. In *Engineering and managing software requirements*, pages 19–46. Springer, 2005.