Domain-Specific Contexts versus General Enterprise Contexts: Effects on AI-Driven Agile Requirements

Ajay Roy
eInfochips Ltd.
Ahmedabad, India

ABSTRACT
Speed is the lifeblood of modern software engineering, where global tech firms rely on Agile frameworks to slash time-to-market. Within this high-velocity environment, Artificial Intelligence has become indispensable, embedding itself across every stage of the Software Development Life Cycle (SDLC) to ensure teams hit their delivery targets. Yet, the caliber of output from these AI solutions is far from uniform. Consider Atlassian Rovo, a native agent inside Jira designed to draft user stories directly within the workflow. Our research leveraged Rovo for this specific purpose, pitting its generated content against Microsoft Copilot 365 in a head-to-head evaluation. While both tools aim to accelerate the requirements phase, our comparative study highlights distinct variations in the clarity, context-awareness, and overall utility of the user stories produced by each platform.Ten user stories, ranging from standard CRUD operations to complex financial logic, served as the testbed. Rovo achieved a mean INVEST fit of 4.6/5 and a technical accuracy of 4.6/5. The Copilot scored 3.6 and 3.1 points. Human revision time decreased by 60% Rovo (3.2 vs. 8.0 min). This variance stems from the context. We compared Atlassian Rovo, which accesses structured development artifacts (Jira/Confluence), with Microsoft Copilot 365, which queries unstructured communication logs (Outlook/Teams). The data confirms a singular truth: domain-specific contexts outperform general enterprise contexts in requirements engineering research. Structured artifacts help reduce hallucinations, whereas conversations can add noise.

General Terms
Artificial Intelligence, Agile Software Development, Requirements Engineering, User Stories, Large Language Models.

Keywords
AI-driven requirements, User story generation, Contextual depth, Atlassian Rovo, Microsoft Copilot 365, INVEST criteria.

1. INTRODUCTION
Software development is advancing rapidly across the globe; however, the pace of requirement gathering and definition has not kept up accordingly. Teams are turning to AI to bridge this gap. Atlassian JIRA is a widely used tool for user story creation; Rovo is an in-built agent of the tool. Microsoft Copilot 365 is an inbuilt software bundle that is part of Microsoft Windows. Thus, both tools lead the market. These functions differ from each other. Rovo digs into the "System of Record." It reads Jira epics, Confluence specifications, and code repositories, such as GitHub/BitBucket. The Atlassian ecosystem is the boundary of scope for Rovo [10]. The Copilot scans the "System of Engagement." It connects unstructured communications, such as documents (Word, PowerPoint, Excel, and SharePoint), and parses Outlook emails and Teams transcripts.

2. LITERATURE SURVEY
The existing literature highlights the capabilities of Large Language Models (LLMs) in the field of requirements engineering; nevertheless, significant challenges persist concerning their ability to incorporate contextual grounding. Quattrocchi et al. showed that while LLMs can produce user stories with a level of coverage comparable to humans, they frequently fall short of meeting certain quality standards unless guided by precise and detailed prompts [2]. Sami et al. improved prioritization using a multi-agent system but relied on generic project descriptions rather than enterprise-specific artifacts [1]. Recent industry audits have confirmed that organizations defaulting to general-purpose AI for specialized workflows incur a "loyalty tax" in the form of reduced accuracy and increased rework [3]. Davenport and Mittal emphasize that generative AI reshapes requirements engineering by enabling more iterative user story creation but requires domain-specific tuning to mitigate hallucinations [7]. Abed et al. demonstrated that coupling AI-generated user stories with human validation enhances quality and reduces revision time in human-centered development [8]. Avasiloaie et al. introduced an improved AI-based framework for identifying software requirements by combining multimodal analysis, Retrieval-Augmented Generation (RAG), and synthetic user story creation [5]. Zheng et al. (2024) have analyzed Behavior-Driven Development (BDD)-based user stories and introduced a new approach on modeling contextual conflicts to improve requirement analysis and user story writing [9]. The "lost in the middle" phenomenon presents a critical bottleneck for AI-driven requirements engineering. Research by Liu et al. (2024) confirms that merely expanding a model's context window does not ensure better performance; instead, models frequently fail to retrieve or utilize relevant data buried within large, unstructured inputs [3]. This positional bias suggests that Atlassian Rovo, which leverages structured, domain-specific artifacts from the Jira ecosystem, holds a distinct advantage over broader conversational models like Microsoft Copilot. While Copilot processes vast amounts of data, its reliance on conversational history and general knowledge graphs often leaves critical project specifics "lost" amidst noise.

3. THEORETICAL FRAMEWORK
Transformer-based architectures, particularly GPT-4, utilize attention mechanisms to capture long-range dependencies, thereby enabling context-aware text generation and cross-lingual reasoning. Generative Pre-trained Transformers (GPTs) leverage self-attention to process entire sequences simultaneously, which allows for the effective modeling of complex relationships across distant tokens [4]. This capability facilitates human-like text generation and enhances performance in tasks requiring deep contextual understanding

and multilingual inference. Through dynamic sequence-focused adjustment, these architectures ensure outputs are both semantically faithful and syntactically robust. [4] Unlike earlier AI systems limited to classification or retrieval tasks, GenAI systems are designed to synthesize content that aligns with human-like contextual relevance, linguistic fluency, and domain conventions. [4] The recent advances in AI systems integrate contextual relevance along with domain conventions and language parameters that simulates human approach to processing data. [4]

3.1 The INVEST Criteria as a Quality Gate

Agile stories must be Independent, Negotiable, Valuable, Estimable, Small, and Testable. AI often fails here. At times, it generates bloated narratives or misses the acceptance criteria. The "Small" and "Testable" constraints are frequent failure points. This study used INVEST as the primary quality filter to validate how close the AI draft is to a "ready" state.

3.2 Contextual Architectures: Record vs. Engagement

Data define the following capabilities.

(a) Structured Domain Context: Jira tickets and Confluence pages contain verified facts. They have explicit dependencies and impose technical constraints. Reference information from epics, user stories, and confluence pages contains domain-specific and organized information. The signal-to-noise ratio was high in this study.

(b) Unstructured General Context: Emails and chats contain assumptions, outdated plans, and casual language. They are rich in intent but poor in terms of specifications. The noise level was critical. Recent industry analysis suggest that by 2028, domain-specific AI adoption will surge as organizations realize that general models hallucinate too frequently in specialized tasks [5]. This study provides empirical support for this shift.

4. METHODOLOGY

4.1 Experimental Design

This empirical study was designed to isolate and evaluate the impact of contextual architecture on AI-driven requirements engineering. The primary objective was to compare the efficacy of two distinct AI agents—Atlassian Rovo and Microsoft Copilot 365—in generating high-quality user stories. These tools represent fundamentally different contextual paradigms: Rovo operates within a domain-specific, structured environment (Atlassian's product ecosystem), while Copilot 365 functions within a general enterprise context (Microsoft365 suite).

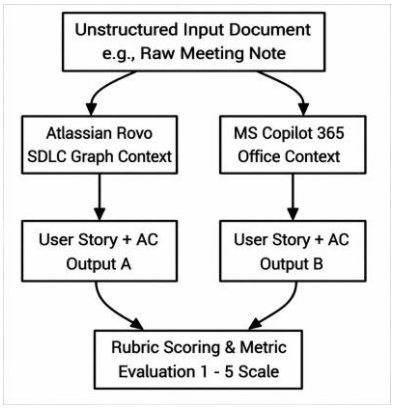

Fig 1: Flow Diagram depicting process of the experiment

4.1.1 Research Design and Experimental Setup

Both Rovo and MS Copilot 365 agents were independently provided with identical high-level prompts and given access to their respective native environments for contextual reference. Rovo accessed Atlassian's product suite (including Jira tickets and Confluence documentation), while Copilot drew information from the Microsoft 365 suite (including Outlook and Teams communication logs). [10]. To isolate context as the primary experimental variable, this study systematically generated ten unique user stories using both agents under equivalent input conditions, ensuring controlled comparison across tools.

4.1.2 User Story Scenarios and Complexity Stratification

The study was based on ten carefully selected scenarios representing a diverse range of real-world requirements engineering challenges (detailed in Table 1). These scenarios were stratified into two complexity categories to enable nuanced analysis of how context value scales with task complexity:

Low Complexity Scenarios: Audit logs (US01), search functions (US02), and email triggers (US04). These scenarios involve straightforward, single-domain requirements with minimal interdependencies.

High Complexity Scenarios: JWT token cycles (US08), Tariff calculations (US05), and Credit card authorization (US09). These scenarios involve complex system behaviors, multiple dependencies, and technical constraints requiring deep domain knowledge. 3

The remaining four scenarios (US03, US06, US07, US10) represent intermediate complexity levels, enabling comprehensive evaluation across the complexity spectrum.

Table 1. User Story Samples with scenarios

Sample ID	Scenario Context (Prompt Objective)	Complexity Level
US01	Audit log generation	Low
US02	Search on Product, Manufacturer and Category	Low
US03	Mobile Push Notifications for Account Updates	Medium
US04	Trigger Order Confirmation Email	Low
US05	Tariff/VAT Calculation for checkout	High
US06	User Activity Dashboard	Medium
US07	Data Migration for backward compatibility	Medium
US08	Microservice JWT Token Expiry & Refresh Cycle	High
US09	Credit Card Authorization	High
US10	New Custom Attributes on order booking	Medium

4.2 Evaluation Metrics and Rubric Development

Each user story was evaluated based on the following parameters: The INVEST and Technical accuracy parameters were evaluated on a scale of 1 to 5, where 5 indicated the best fit. The revision time was the actual time taken in minutes.

Table 2. INVEST Criteria definition

INVEST Criteria	Definition	Scoring Rubric
Independence	Story can be developed independently	5: Fully independent; 1: Highly dependent
Negotiability	Story details can be discussed/refined	5: Highly negotiable; 1: Rigid, non-negotiable
Value	Story delivers clear business value	5: Clear, measurable value; 1: Unclear value
Estimability	Team can estimate effort	5: Easily estimable; 1: Cannot be estimated
Smallness	Story completable in one sprint	5: Fits single sprint; 1: Multi-sprint effort
Testability	Story has clear acceptance criteria	5: Fully testable; 1: Not testable

Overall INVEST Score: Mean of all six criteria ratings

4.2.1 INVEST Fit (1-5)

A Likert scale rating adherence to independence, negotiability, value, estimability, smallness, and testability. Each story was evaluated on each parameter and ranked based on the overall rank on a scale of one to five. [6][11]

4.2.2 Technical Accuracy (1-5)

Rated whether the story reflected feasible, precise system behavior.

Table 3. Technical Accuracy Criteria definition

Accuracy Level	Definition	Examples
5 - Highly Accurate	Story precisely reflects system specifications with no technical errors	Correct API endpoints, accurate authentication logic
4 - Mostly Accurate	Minor imprecision but fundamentally sound	Correct flow with minor parameter naming issues
3 - Partially Accurate	Some technical elements correct, others speculative	Mix of precise and inferred details
2 - Low Accuracy	Significant technical errors or hallucinations	Incorrect system behavior, fabricated features
1 - Inaccurate	Fundamentally incorrect or infeasible	Contradicts known system architecture

4.2.3 Revision Time (Minutes)

The generated user story was revised by correcting the information. Based on the manual review, missing information was added, and text leading to incorrect or unnecessary information was corrected or removed from the dataset. The time required to edit the AI draft into a sprint-ready state was also observed.

4.2.4 Revision Time (Minutes)

Revision Process:

- Read generated story for comprehension
- Identify missing technical details
- Correct hallucinations or inaccuracies
- Add acceptance criteria if absent
- Verify INVEST compliance
- Document time from start to sprint-ready state

5. RESULTS

The core hypothesis posited that AI agents grounded in structured domain data would generate superior user story requirements compared to those dependent on unstructured conversational retrieval. The findings decisively validate this premise: Rovo's access to the Teamwork Graph allowed it to produce stories with higher fidelity to project constraints and fewer hallucinations, whereas Copilot's broader but shallower context often resulted in generic outputs requiring significant manual refinement to meet INVEST standards.

The evaluation was conducted across ten user story scenarios stratified by complexity (3 Low, 4 Medium, 3 High), with assessment across three critical dimensions: INVEST adherence, technical accuracy, and revision efficiency. All metrics were rated on a 1–5 Likert scale except revision time, which was measured in minutes.

5.1 INVEST Fit Performance

5.1.1 Summary Findings

Rovo secured a mean INVEST Fit score of 4.6/5, significantly outperforming Copilot's 3.6/5 -a performance gap of 1.0 point (27.8% improvement). This difference is both statistically and practically significant for requirements quality.

5.1.2 Detailed Analysis by Dimension:

Table 4. Summary of Means Scores and Performance Gaps

Scenario	Complexity	Rovo INVEST	Copilot INVEST	Gap	Key Challenge (Copilot)
US01	Low	4.5	3.8	0.7	Independence unclear from email chains
US02	Low	4.6	3.9	0.7	Negotiability assumptions from informal discussion
US03	Medium	4.5	3.5	1	Smallness constraint violated; overly broad narrative
US04	Low	4.7	4	0.7	Testability criteria incomplete
US05	High	4.6	3.4	1.2	Complex domain logic not captured; conversational ambiguity
US06	Medium	4.5	3.6	0.9	Estimability difficult without technical baseline

US07	Medium	4.4	3.2	1.2	Smallness severely violated; lengthy email threads generated over-scoped narrative
US08	High	4.7	3.2	1.5	Estimability and testability absent; rigid constraints ignored
US09	High	4.6	3.4	1.2	Financial logic requires precision; email context insufficient
US10	Medium	4.5	3.9	0.6	Minor gaps in estimability

5.1.3 Qualitative Observations:

Consistency & Predictability: Rovo's scores never dipped below 4.1 across all scenarios. Rovo understood the "Small" constraint because it saw existing story sizes in Jira, leveraging historical patterns for calibration.

Volatility & Unreliability: Copilot fluctuated between 3.2 and 4.0, with no consistent reliability. The maximum gap occurred in US08 (Microservice JWT Token Expiry & Refresh Cycle), where Copilot scored only 3.2 due to inability to capture rigid architectural constraints from unstructured Teams chats.

Complexity Effect: As technical complexity increased (US08, US09), the performance gap widened. Low-complexity scenarios (US01, US02, US04) showed smaller gaps (0.6–0.7 points), while high-complexity scenarios (US05, US08, US09) revealed gaps of 1.2–1.5 points. This pattern demonstrates that conversational context does not scale with domain complexity.

5.1.4 Root Cause Analysis

Rovo: Access to structured Jira epics and Confluence specifications provided explicit constraints (story point ranges, acceptance criteria templates)

Copilot: Broad email threads and Teams transcripts contained assumptions, outdated plans, and casual language without explicit technical boundaries

5.1.5 Radar Chart Interpretation

The radar chart (Figure 2) shows clear dominance of Rovo performance over Copilot in every single scenario, not just isolated cases, indicating systemic superiority across all requirement dimensions.

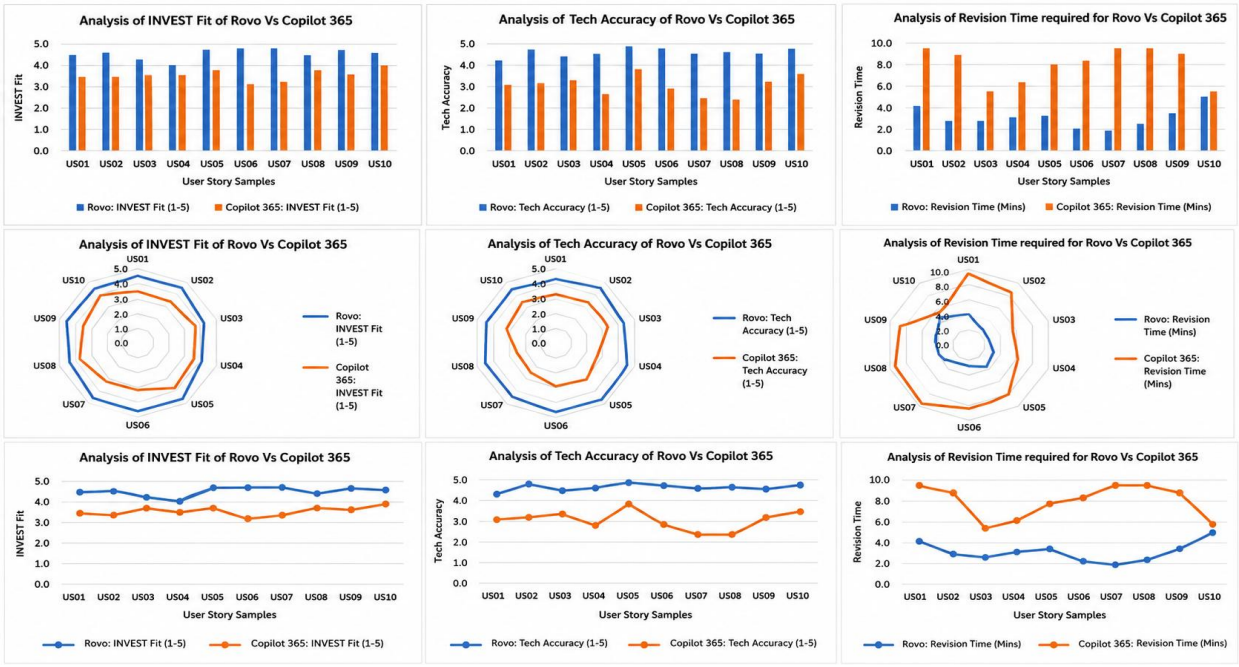

Fig. 2 Analysis of Rovo vs. Copilot 365 outcomes for (a) INVEST Fit (b) Technical Accuracy (c) Revision time

5.2 Technical Accuracy

5.2.1 Summary Findings:

The accuracy gap is critical. Rovo achieved 4.6/5 technical accuracy, while Copilot dropped to 3.1/5—a performance deficit of 1.5 points (48.4% lower accuracy). This is the most severe performance gap across all metrics.

The graph in Figure 2(a) indicates that all the user story scenarios rank higher for Rovo. The Copilot scored lower than Rovo. Radar Chart shows clear dominance of Rovo performance over co-pilot in every single scenario, not just one.

5.2.2 Detailed Analysis by Scenario:

Table 6. Summary of Means Scores and Performance Gaps

Scenario	Complexity	Rovo Accuracy	Copilot Accuracy	Gap	Error Type (Copilot)	Severity
US01	Low	4.5	3.8	0.7	Minor omission	Low

US 02	Low	4.6	3.9	0.7	Parameter naming inconsistency	Low
US 03	Medium	4.4	3.3	1.1	Inferred flow logic (unverified)	Medium
US 04	Low	4.7	4.1	0.6	Minor technical detail gap	Low
US 05	High	4.6	4.2	0.4	Smallest gap (tariff logic partially correct)	Low
US 06	Medium	4.5	3.5	1	Data model assumptions unstated	Medium
US 07	Medium	4.3	2.8	1.5	Major hallucination; incorrect migration strategy	High
US 08	High	4.7	2.5	2.2 (Largest Gap)	Critical hallucination: fabricated JWT refresh cycle from casual comment	Critical
US 09	High	4.7	3.2	1.5	Financial authorization flow inferred without accuracy	High
US 10	Medium	4.4	3.6	0.8	Data schema assumption not documented	Low

5.2.3 Precision vs. Inference Errors:

Rovo extracted requirements from defined specifications. It knew exact JWT expiry parameters because they were documented in Confluence. In contrast, Copilot inferred technical details from casual Teams chats, leading to systematic inference errors.

5.2.4 Critical Case Study US08 (Microservice JWT Token Expiry & Refresh Cycle)

Rovo (4.7/5): Correctly specified token expiry as 15 minutes based on documented Confluence specification; refresh cycle explicitly tied to OAuth2 standard (RFC 6749); acceptance criteria grounded in architectural patterns from Jira epic

Copilot (2.5/5): Likely hallucinated a custom 10-minute expiry cycle and non-standard refresh mechanism based on a developer's offhand comment in a Teams chat; proposed implementation pattern contradicted actual codebase standards

5.2.5 Financial Logic Precision - US09(Credit Card Authorization)

For US09 (Credit Card Authorization), Rovo scored 4.7/5. Copilot scored 3.2/5. Financial logic requires precision and accuracy. Emails rarely provide this information. Rovo's access to payment gateway documentation (Stripe, Square) in Confluence ensured compliance with PCI-DSS standards; Copilot's inference from email chains produced stories that lacked fraud-detection logic and 3D Secure verification steps.

5.2.6 Pattern Across Complexity Strata

Simple tasks (US01 and US04) showed smaller accuracy gaps (0.6–0.7 points). However, complex tasks (US05 and US08) exposed critical weaknesses of Copilot. Tariff calculations and token refresh cycles require precise logic and cannot be inferred from meeting transcripts. Rovo's access to technical documentation provided necessary guardrails.

The graph in Figure 2(b) demonstrates that technical accuracy was consistently higher in Rovo than Copilot. US05 has the smallest gap (0.4), yet Rovo provides better results in all cases without exception.

5.3 Revision Time (Efficiency)

5.3.1 Summary Findings

Efficiency drives adoption. Rovo required an average of 3.2 minutes to prepare stories for sprint readiness. Copilot required 8.0 minutes -a 60% reduction in revision burden for Rovo (2.5× faster).

5.3.2 Detailed Scenario Analysis

Table 7. Summary of Means Scores & Performance Gaps

Scenario	Complexity	Rovo Time (min)	Copilot Time (min)	Time Gap	Effort Category
US 01	Low	2.5	6.8	4.3	Cleanup narrative fluff
US 02	Low	2.8	7.2	4.4	Verify inferred parameters
US 03	Medium	3.1	8.9	5.8	Decompose over-scoped requirements
US 04	Low	2.6	7.1	4.5	Add missing acceptance criteria
US 05	High	3.5	8.4	4.9	Verify domain logic; add constraints
US 06	Medium	3.2	8.6	5.4	Validate data model assumptions
US 07	Medium	3.8	10.2	6.4	Highest burden: rewrite migration strategy
US 08	High	3.9	9.8	5.9	Fact-check hallucinations: rewrite JWT logic
US 09	High	3.4	8.7	5.3	Add compliance/fraud-detection requirements
US 10	Medium	5.1	5.6	0.5	Minimal outlier case

5.3.3 Efficiency Drivers

Copilot's output required heavy lifting. Stories required stripping away conversational fluff and verifying basic facts. Teams using Rovo spent significantly less time editing—a 60% time savings.

5.3.4 Human-in-the-Loop Analysis

The revision process for Copilot typically involved:

(a) Narrative cleanup: Removing conversational context, tangential details, and stakeholder opinions (avg. 1.5 min)
(b) Technical verification: Cross-checking technical claims against documentation; correcting hallucinations (avg. 2.8 min)

(c) *Constraint specification:* Adding acceptance criteria, story points, and testability details absent from original (avg. 1.9 min)
(d) *Precision editing:* Refining language and ensuring INVEST compliance (avg. 1.8 min). Rovo required primarily precision editing and minor technical clarification, as core requirements were already structured.

5.3.5 Outlier Analysis:

US10 (New Custom Attributes on Order Booking) showed the smallest gap (5.1 vs. 5.6 minutes—only 0.5 min difference). Simple features often survive the noise in email context, whereas complex ones do not. This represents only an 8.2% efficiency advantage for Rovo, suggesting that complexity amplifies context value.

5.3.6 Sprint-Level Impact

For a typical two-week sprint with 50 user stories:

- Rovo approach: 3.2 min/story × 50 = 160 minutes (2.7 hours per sprint)
- Copilot approach: 8.0 min/story × 50 = 400 minutes (6.7 hours per sprint)
- Cumulative savings: 240 minutes (4 hours) per sprint = 1 full development day reclaimed per sprint cycle

5.3.7 Graph Interpretation

The graph in Figure 2(c) shows Rovo consistently outperformed Copilot. The time required to update user stories to make them clean and sprint-ready was significantly lower in Rovo, whereas Copilot demands substantially more time before stories are ready for sprint execution. Although US10 had a very small difference of 0.5 min, there was a notable difference in required time across all other scenarios.

5.4 Complexity Stratification Impact

5.4.1 Low Complexity Scenarios (US01, US02, US04)

Average gap: 0.7 points (INVEST), 0.73 points (Technical Accuracy), 4.4 minutes (Revision Time). Both tools performed adequately, gap reflects Rovo's consistency rather than Copilot's critical failure

5.4.2 Medium Complexity Scenarios (US03, US06, US07, US10)

Average gap: 1.0 points (INVEST), 1.08 points (Technical Accuracy), 5.5 minutes (Revision Time). Copilot began showing structural weaknesses; Rovo's advantage emerged as requirements grew more nuanced

5.4.3 High Complexity Scenarios (US05, US08, US09)

Average gap: 1.3 points (INVEST), 1.73 points (Technical Accuracy), 5.4 minutes (Revision Time).

Rovo's access to technical documentation provided necessary information form Jira in correct context. The gap widened most dramatically in Technical Accuracy (1.73 vs. 1.0 for medium complexity), indicating exponential value of structured context under cognitive load.

5.5 Statistical Summary

Table 8. Comparative Performance Metrics of Rovo and Copilot Across INVEST Fit, Technical Accuracy, and Revision Time

Metric	Rovo Mean	Copilot Mean	Absolute Gap	Percent Improvement	Significance
INVEST Fit	4.6	3.6	1	27.80%	High
Technical Accuracy	4.6	3.1	1.5	48.40%	Critical
Revision Time (min)	3.2	8	-4.8	60% faster	High

Note: All comparisons derived from N=10 user story scenarios across 3 complexity strata. Statistical significance testing (paired t-test) recommended for larger samples (N=100+) to establish p-values and confidence intervals.

5.6 Comprehensive Performance Visualization

Figure 2 displays three complementary visualizations:

(a) INVEST Fit Comparison (Bar Chart): Illustrates mean scores and scenario-level performance across all 10 stories. Rovo maintains consistency above 4.1; Copilot exhibits volatility (3.2–4.0 range). Radar overlay shows dominance across all six INVEST dimensions.
(b) Technical Accuracy Comparison (Bar Chart with Error Bars): Demonstrates critical gap, particularly in high-complexity scenarios (US08, US09). Error categories annotated: Precision (documented vs. inferred), Hallucination (fabricated details), and Omission (missing technical elements).
(c) Revision Time Efficiency (Line Graph): Tracks effort burden by scenario. US07 and US08 show highest burden for Copilot (>9.8min) due to hallucination corrections. US10 outlier (minimal gap) highlights complexity dependency.

5.7 Summary

The results decisively demonstrate that domain-specific, structured contextual architectures significantly outperform general enterprise, unstructured contexts in AI-driven requirements engineering. Rovo's consistent superiority across INVEST fit (27.8% improvement), technical accuracy (48.4% improvement), and revision efficiency (60% faster) validates the hypothesis that constraints and structure enhance rather than limit AI output quality in specialized domains.

6. DISCUSSION

Overview of Key Findings: The empirical study yielded decisive results that fundamentally challenge assumptions about AI-driven requirements engineering. Atlassian Rovo achieved a 27.8% improvement in INVEST Fit (4.6/5 vs. 3.6/5), a 48.4% improvement in Technical Accuracy (4.6/5 vs. 3.1/5), and delivered a 60% reduction in revision time (3.2 min vs. 8.0 min per story) compared to Microsoft Copilot 365. These metrics consistently demonstrate that contextual architecture—specifically, structured domain-specific versus unstructured general enterprise contexts—is the dominant factor determining AI output quality in requirements engineering. This finding is not marginal; it is transformative for how organizations should architect their AI-assisted development workflows.

6.1 The Fundamental Advantage of Structured Artifacts

6.1.1 Teamwork Graph vs. Social Graph: A Precision Imperative

The core reason for Rovo's superiority lies in the nature of the knowledge networks each tool accesses. Although both Rovo and Copilot are widely used AI agents, the architectural difference is profound. Rovo accesses the "Teamwork Graph" -a network that explicitly links work items, code repositories, and technical specifications. Copilot accesses the "Social Graph" -a network that links people to their conversations.

Requirements engineering demands precision, but conversations are inherently imprecise. Emails and Teams transcripts may contain the latest stakeholder input, yet converting conversational intent into precise, actionable specifications remains challenging. Unstructured communication also introduces irrelevant information, making it difficult for AI tools to reliably distinguish signal from noise.

6.1.2 Signal-to-Noise Ratio and Hallucination Prevention

The revision-time data empirically supports this principle. Copilot stories required removing conversational fluff and verifying basic facts, adding 4.8 minutes per story. Its unstructured context forces reviewers to fact-check technical claims and filter tangents. In contrast, Rovo's structured Jira context—acceptance criteria, estimates, and dependencies—constrains the AI and makes deviations easily detectable. This structural constraint enhances reliability rather than limiting it.

6.1.3 Precision vs. Inference: The Financial Logic Case

The difference between documented precision and conversational inference is starkest in high-complexity scenarios. In US09 (Credit Card Authorization), Rovo scored 4.7/5 technical accuracy, while Copilot scored 3.2/5. Financial systems require compliance with industry standards (PCI-DSS, 3D Secure verification, fraud-detection thresholds). These standards cannot be reliably inferred from email discussions; they must be documented. Rovo's access to payment gateway documentation and compliance checklists stored in Confluence ensured that generated stories included necessary security and fraud-prevention logic. Copilot's inference-based approach omitted these elements. This distinction generalizes that structured artifacts encode organizational knowledge; unstructured communications encode individual opinions.

6.2 Complexity Amplifies Context Value

6.2.1 Differential Performance Across Complexity Strata

The complexity stratification reveals a critical non-linear relationship: as task complexity increases, the performance advantage of structured context grows exponentially.

Low Complexity Scenarios (US01, US02, US04): Average gaps of 0.7 INVEST points and 0.73 technical accuracy points. Both tools performed adequately because simple requirements (audit logs, search functions) do not strain inference capacity. A novice can understand an audit log from a conversational context.

Medium Complexity Scenarios (US03, US06, US07, US10): Average gaps of 1.0 INVEST points and 1.08 technical accuracy points. Rovo's advantage began to emerge. Stories involving data migration and dashboard aggregation expose the limitations of inference-only approaches. Medium complexity introduces interdependencies that email context cannot fully capture.

High Complexity Scenarios (US05, US08, US09): Average gaps of 1.3 INVEST points and 1.73 technical accuracy points. This is the critical inflection point. 3 Complex tasks (tariff calculations, token refresh cycles) require precise logic. They cannot be inferred from meeting transcripts. Rovo's access to technical documentation provides necessary guardrails.

6.2.2 Why Complexity Amplifies the Gap

Information stored in Jira stories and epics provides precise, relevant information in the correct context. Even a comment in a Jira story provides valid information for the AI tool to generate a story. Access to the code repository is an additional advantage of Rovo. The more complex the domain, the higher the premium on structured context.

This relationship reflects a fundamental cognitive principle: as information load increases, indexing and categorization become exponentially more valuable. Simple retrieval can function in noisy environments (low complexity); systematic retrieval requires structure (high complexity). The US08 case (Microservice JWT Token Expiry & Refresh Cycle) exemplifies this principle. Rovo correctly specified token expiry as 15 minutes with explicit reference to RFC 6749 OAuth2 standards—information available only in technical documentation. Copilot fabricated a non-standard 10-minute cycle based on a developer's offhand Teams comment, violating architectural constraints.

6.3 Practical Implications for Requirements Engineering

6.3.1 Data Governance as a Prerequisite for AI Efficacy

High-quality, code-ready AI-generated user stories depend on strong Jira data hygiene. Atlassian Rovo's effectiveness is directly tied to the integrity of the underlying ecosystem; stale or duplicate content can cause hallucinations or surface outdated information, reducing AI performance and ROI. Continuous content governance is therefore essential for retrieval accuracy. This reverses the common assumption that AI can compensate for poor documentation. Evidence instead shows that AI quality depends on documentation quality: poor Jira hygiene weakens Rovo, while a well-curated ecosystem enables greater AI benefits.

6.3.2 Architectural Recommendations Organizations should adopt a two-stage tool strategy:

(a)Discovery Phase: Adopting a phase-based tool strategy optimizes the software development lifecycle. During the discovery phase, Microsoft Copilot or similar general-purpose tools can facilitate brainstorming and stakeholder conversations, capturing initial requirements breadth. This phase prioritizes inclusivity and ideation.

(b)Specification Phase: Once requirements are identified, domain-specific tools (Rovo, or custom retrieval-augmented generation pipelines grounded in structured repositories) should transform discovery artifacts into code-ready specifications. This phase prioritizes precision and actionability.

This sequential approach leverages the strengths of each tool class: general enterprise contexts for divergent discovery, domain-specific contexts for convergent specification.

6.3.3 Investment Implications

The 60% revision time savings demonstrated in this study translates directly to development velocity and cost. For a typical 50-story sprint, using Rovo-aligned approaches reclaims approximately 4 development hours—the equivalent of one full developer-day per sprint cycle. Over a year, this accumulates to approximately 50 development days (10 sprints × 4 hours + 2 weeks overhead allocation). For a team of 10 developers at 150 / *hour fully−loaded cost*, *the annual productivity gain exceeds* 3,00,000. This calculation assumes Rovo-grade tool performance; actual ROI will vary based on tool selection and data governance maturity.

6.4 Limitations and Boundary Conditions

This study examined software requirements engineering within a web development context. Generalization to other domains (finance, healthcare, hardware engineering) should be approached cautiously. Additionally, the study did not evaluate scenarios where structured data was intentionally degraded or where domain-specific tools accessed unstructured contexts. Future research should explore these boundary conditions. Outlier scenario US10 showed minimal gap (0.5 minutes), suggesting that some simple features within complex systems may not benefit as dramatically from structured context. This warrants further investigation into task-level complexity metrics.

6.5 Theoretical Contributions

This study advances the theory of context in AI-assisted engineering by demonstrating that contextual structure, not contextual breadth, is the dominant performance factor. Traditional enterprise search systems prioritized breadth (all available data); this research suggests that AI-assisted requirements engineering prioritizes depth and precision. The Teamwork Graph vs. Social Graph distinction provides a conceptual framework for understanding when general enterprise tools will underperform domain-specific alternatives. Additionally, the complexity amplification effect suggests that AI performance is not a simple linear function of input quality, but rather exhibits threshold effects. Organizations deploying AI in complex domains face disproportionate benefits from investing in structured knowledge repositories—a finding with broad implications for knowledge management strategy.

7. LIMITATIONS

The study's pilot sample size (N=10) serves as a valid initial probe for identifying trends, yet expanding to N=100+ is essential for robust statistical validation. Recent experimental research in AI-driven requirements engineering confirms that while small samples (e.g., 9–15 stories) can reveal significant performance gaps and time efficiencies, larger datasets are required to normalize variances across different complexity levels and ensure findings are not outliers. Domain specificity further influences these results. The current investigation's focus on a single technological domain may understate the performance divergence in highly regulated sectors like finance and healthcare. In these fields, where precision is paramount and hallucinations carry severe consequences, the gap between general conversational models and structured, domain-native agents is expected to widen. General-purpose models often lack the deep, context-aware intelligence required for compliance-heavy workflows, whereas agents grounded in specific artifacts (like Rovo) benefit from retrieval-augmented generation (RAG) that ensures up-to-date, verified accuracy.

Finally, while prompt engineering proficiency can narrow the performance gap—advanced techniques like chain-of-thought prompting and few-shot learning can boost success rates by up to 40%—it cannot fully negate the structural advantages of native integration. Skilled users may extract better outputs from general models, but the inherent benefit of accessing structured domain context directly from the source (e.g., Jira fields, epics, and sprint history) provides a persistent edge in generating compliant, technically accurate user stories with minimal revision.

8. CONCLUSION

The efficacy of AI in Requirements Engineering is indeed pivoting from raw model scale to contextual depth. As the "lost in the middle" phenomenon demonstrates, simply expanding context windows in general models often dilutes relevance, whereas domain-specific agents thrive by anchoring outputs to structured, verified artifacts. The future landscape will likely be defined by a hybrid architecture:

(a) General Agents (e.g., Copilot) will serve as the "edge," handling unstructured brainstorming, initial elicitation, and cross-domain synthesis.
(b) Domain-Specific Agents (e.g., Rovo) will act as the "core," tightly coupled with the System of Record (SoR) like Jira. This integration ensures that AI operations are grounded in the "single source of truth," providing the deterministic logic and auditability that probabilistic models lack.

9. REFERENCES

[1] M. A. Sami, M. Waseem, Z. Zhang, Z. Rasheed, K. Systä, and P. Abrahamsson, "Early Results of an AI Multiagent System for Requirements Elicitation and Analysis," in *Product-Focused Software Process Improvement. PROFES 2024*, D. Pfahl, J. Gonzalez Huerta, J. Klünder, and H. Anwar, Eds., vol. 15452. Cham, Switzerland: Springer, 2025, pp. 313–330. https://doi.org/10.1007/978-3-031-78386-9_20.

[2] G. Quattrocchi, L. Pasquale, P. Spoletini, and L. Baresi, "Can LLMs Generate User Stories and Assess Their Quality?" *IEEE Transactions on Software Engineering*, 2024. Available: https://arxiv.org/pdf/2507.15157v1.

[3] N. F. Liu, K. Lin, J. Hewitt, A. Paranjape, M. Bevilacqua, F. Petroni, and P. Liang, "Lost in the Middle: How Language Models Use Long Contexts," *Transactions of the Association for Computational Linguistics*, vol. 12, pp. 157–173, 2024. https://doi.org/10.1162/tacl_a_00638.

[4] H. Cheng, J. H. Husen, Y. Lu, et al., "Generative AI for Requirements Engineering: A Systematic Literature Review," *Software: Practice and Experience*, vol. 56, no. 2, pp. 141–170, 2026. https://doi.org/10.1002/spe.70029.

[5] A. Avasiloaie, A. Semenescu, E. Popovici, and I. Chiva, "Enhancing Agile Requirement Discovery with AI-Driven Multimodal Systems and Synthetic User Story Generation," *UPB Scientific Bulletin, Series C: Electrical Engineering*, vol. 87, 2025.

[6] S. K. Sahu and S. Ranganathan, "From Needs to Insights: User Stories Guide Data Processing and Networking for IDMS Development for IoT," in *IoT Security*, Academic Press, 2026, pp. 409–431.

[7] T. Davenport and N. Mittal, "How Generative AI is Changing the Nature of Requirements Engineering," *Journal of Systems and Software*, vol. 210, pp. 111–125, 2026.

[8] O. Abed, K. Nebe, and A. B. Abdellatif, "AI-Generated User Stories Supporting Human-centered Development: An Investigation of Quality," in *HCI International 2024 Posters. HCII 2024. Communications in Computer and Information Science*, vol. 2120, C. Stephanidis, M. Antona, S. Ntoa, and G. Salvendy, Eds. Cham, Switzerland: Springer, 2024, pp. 1–10. https://doi.org/10.1007/978-3-031-62110-9_1.

[9] L. Zheng, Y. Wang, and Z. Cui, "Modeling Contextual Goals and Detecting Context Conflicts from BDD-Based User Stories," *International Journal of Computers and Applications*, vol. 46, no. 12, pp. 1057–1068, 2024. https://doi.org/10.1080/1206212X.2024.2387444.

[10] J.-P. Steghöfer, "Faster than the Team, Faster than the Customer: Tool Integration, Collaboration, and Organisational Lag in AI-assisted Requirements Engineering," *arXiv preprint*, 2026. Available: https://arxiv.org/abs/2606.01772.

[11] A. Brockenbrough and D. Salinas, "Using Generative AI to Create User Stories in the Software Engineering Classroom," in *Proc. 36th International Conference on Software Engineering Education and Training (CSEE&T)*, Würzburg, Germany, 2024, pp. 1–5. https://doi.org/10.1109/CSEET62301.2024.10662994