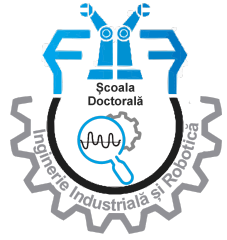




MINISTRY OF EDUCATION AND RESEARCH
National University of Science and Technology
POLITEHNICA Bucharest

Doctoral School of
Industrial Engineering and Robotics



Andra-Paula AVASILOAIE

PHD THESIS

**Contributions to the development
of AI-driven approaches for
software requirement discovery
and user story generation**

Scientific coordinator,
prof.univ.habil.dr.ing.ec.mat. Augustin SEMENESCU
(POLITEHNICA Bucharest)

- 2025 -

Table of contents

Introduction	3
Chapter 1. Research Framework	5
Chapter 2. Software Development Ecosystem and Methodologies	6
2.1 Current Software Development Ecosystem	6
2.2 Project Management in Software Development	6
2.3 Software Development Life Cycle (SDLC) Models and Their Evolution	6
2.4 Agile vs. Waterfall: Comparative Perspectives	7
2.5 Agile Principles and Values	8
2.6 Challenges in Managing Software Development Projects	9
Chapter 3. Roles and Responsibilities in Agile Teams	10
3.1 The Role of the Business Analyst	10
3.2 The Role of the Product Owner and the Product Manager	10
3.3 Comparative Scenarios of Role Distribution	10
3.4 Research Findings	11
Chapter 4. Requirements Engineering and User Stories	12
4.1 Sources and Analysis of Requirements	12
4.2 The Concept and Structure of User Stories	13
4.3 Epics, Acceptance Criteria, and Definition of Ready	13
4.4 Limitations of Traditional User Story Writing	13
Chapter 5. Standardization of User Story Writing	15
5.1 Proposed New Standard for User Story Definition	15
5.2 Validation through Practice and Training	15
5.3 Towards Tool-Supported Standardization	16
5.4 From Templates to Automated Generation	16
Chapter 6. The Jira Extension for Deterministic Automation of User Story Writing	17
6.1 Jira as a Platform for Agile Requirements Management	17
6.2 Motivation for Developing the Jira Extension	17
6.3 Design and Functionality of Specs-Assistant	17
6.4 User Interface of the Extension	18
6.5 Benefits Observed in Practice	19

Chapter 7 – AI-powered User Story Generation Interface	20
7.1 Related Work	20
7.2 From Manual Templates to AI-Enhanced Generation	20
7.3 Comparative Evaluation of AI Models	23
7.4 Findings and Challenges	24
Chapter 8. AI-Driven Multimodal Requirement Discovery	25
8.1 Foundational Technologies in AI-Driven Requirement Discovery	25
8.2 From Fragmented Tools to Integrated Discovery Pipelines	26
8.3 Dual-Phase Generation Approach	26
8.4 System Overview	26
8.5 Inference Workflow	29
Chapter 9: A Simplified Iteration of Discovery AI with GPT-5	31
9.1 System Design	31
9.2 Comparative Analysis: Old vs. New Iteration	32
9.3 Interface and Usability	33
9.4 Opportunities and Limitations	35
Chapter 10. Conclusions	36
10.1. Concluzii generale	36
10.2. Original Contributions	36
10.3. Future Research Directions	37
10.4. Work Synthesis	37
References	38

Introduction

The doctoral research undertaken throughout this thesis reflects both the academic motivation and the professional background of the author, with a focus on advancing the field of software requirements engineering through deterministic and AI-driven approaches. Rooted in the challenges experienced as a Business Analyst, the work highlights the critical importance of bridging communication gaps between stakeholders and development teams and of translating ambiguous business inputs into precise, actionable specifications.

The study is motivated by the growing complexity of software development projects in the context of digital transformation and the increasing adoption of Agile methodologies. While Agile has introduced flexibility and collaboration into software development, requirement discovery remains a persistent challenge. User stories, although widely adopted as lightweight artifacts for capturing client needs, often suffer from ambiguity, inconsistency, and formatting overheads. The author's early professional experiences, including training workshops delivered to over seventy aspiring analysts and product owners (with more than 80% later employed in the field), provided empirical evidence of these issues and shaped the direction of the research.

The introduction identifies two key bottlenecks:

1. The uneven distribution of responsibilities across Agile teams depending on the presence or absence of roles such as Business Analyst, Product Owner, or Project Manager.
2. The significant time required to format, structure, and standardize user stories, despite their apparent simplicity.

Initial research explored these challenges by analyzing role distribution in Agile teams and by proposing a new standard for writing user stories. Early experiments revealed measurable improvements in clarity and efficiency but also emphasized the inherent difficulty of extracting requirements directly from unstructured communication channels, such as stakeholder interviews, workshops, and business meetings.

At the same time, advances in Artificial Intelligence, and particularly Large Language Models (LLMs), introduced new possibilities for transforming requirement engineering. Automatic Speech Recognition (ASR), Retrieval-Augmented Generation (RAG), and synthetic data generation now enable multimodal systems that process raw business meeting data and produce structured Agile artifacts. However, most prior work in this area still treats requirements as purely textual artifacts, disregarding the multimodal dimensions of communication (intonation, emphasis, shared context) which strongly influence meaning [1][2].

The thesis is organized into ten chapters, each representing a stage of this research trajectory:

Chapter 1 sets the foundation by presenting the research motivation, objectives, scope, and methodology.

Chapter 2 provides an extensive literature review on requirement engineering, user story structuring, AI in software development, and prompt engineering for LLMs.

Chapter 3 details the research methodology, highlighting the iterative design process and mixed-methods approach combining exploratory design, prototype development, and user feedback.

Chapter 4 presents a series of case studies where deterministic formats and early user story standards were tested in educational and professional settings, validating the initial hypotheses.

Chapter 5 introduces the Extended User Story (EUS) standard, a new model for writing user stories that improves clarity, traceability, and alignment through visual formatting rules and structural conventions.

Chapter 6 describes the development and implementation of a JIRA extension that operationalizes the EUS standard by providing reusable templates and interface elements to support Business Analysts and Product Owners.

Chapter 7 advances the automation process by introducing an AI-powered interface that generates user stories from text prompts using OpenAI's GPT API.

Chapter 8 presents the most complex solution — an AI-driven multimodal requirement discovery system. This chapter details the design, implementation, and evolution of the architecture

Chapter 9 explores a future-oriented simplified version of the Discovery AI architecture, leveraging GPT-5's multimodal capabilities to reduce complexity while maintaining performance.

Chapter 10 concludes the thesis by summarizing the findings, contributions, and future directions.

This progressive journey illustrates the transition from deterministic approaches to advanced AI-driven architectures, demonstrating both academic contributions and practical applicability to industry.

Chapter 1. Research Framework

Requirements engineering is a critical stage of the software development lifecycle (SDLC), yet it often suffers from ambiguity, costly rework, and project overruns [1], with poor requirement management identified as a leading cause of software project failures [2]. Agile methodologies rely on user stories [3], but ensuring clarity, consistency, and well-formulated acceptance criteria remains a persistent challenge [4]. Deterministic tools such as Jira extensions have improved efficiency, saving up to 60% of the effort [5], though they remain domain-specific and fail to capture requirements hidden in unstructured inputs like interviews and meetings [6]. While AI and NLP have shown promise [7], most solutions still reduce requirements to text-only artifacts, neglecting the multimodal nature of communication [8].

This thesis addresses these limitations through five research questions, focusing on team role configurations (RQ1), standardization of user story writing (RQ2), evaluation of deterministic tools (RQ3), design of AI-driven multimodal approaches integrating ASR, RAG, and LLMs (RQ4), and comparative assessment of deterministic versus AI-driven systems (RQ5). The objectives align with these questions, ranging from analyzing team responsibilities to designing and benchmarking both deterministic and AI-based solutions.

The contributions span multiple domains: theoretical (a framework linking Agile roles to requirements and a structured standard for user stories [4]), methodological (surveys, workshops, and case studies), practical (a Jira extension demonstrating measurable efficiency gains [5]), technological (an AI-driven multimodal system integrating ASR, RAG, and LLMs [5]), and empirical (evaluations based on clarity, traceability, and scalability). Methodologically, the work employs a mixed approach—qualitative methods (workshops, surveys, interviews), quantitative metrics (time, clarity, efficiency benchmarks), prototype development, and comparative evaluation of manual, deterministic, and AI-driven approaches.

By bridging academic research and industrial practice, the thesis proposes both conceptual and operational advances for improving requirement discovery and user story generation.

Chapter 2. Software Development Ecosystem and Methodologies

2.1 Software Development Ecosystem

The software development ecosystem has changed profoundly in recent decades, evolving from a narrow technical discipline to a strategic driver of organizational competitiveness. With globalization, distributed teams, and regulatory pressures, software projects today must integrate not only engineering practices but also business strategy, compliance, and user experience. The environment is widely described as VUCA—volatile, uncertain, complex, and ambiguous—requiring adaptive methodologies [9]. The COVID-19 pandemic accelerated this trend, pushing companies to transition rapidly toward digital products and services [10].

Software development encompasses three major categories: system software, programming software, and application software. These domains are increasingly interconnected, and modern projects require the integration of technical and business perspectives. Unlike the past, when software development was primarily about building functional code, it is now part of enterprise-wide digital transformation strategies.

Project management ensures that ideas, people, and resources are aligned to deliver value. The classical lifecycle of initiation, planning, execution, monitoring, and closure remains relevant, but its rigid sequence has been challenged by the realities of volatile environments. Agile practices have introduced iteration, feedback loops, and continuous adaptation, which make projects more resilient. Still, recurring issues persist: scope creep caused by shifting requirements, the difficulty of balancing speed and quality within budget constraints, and the need to coordinate distributed teams effectively. New categories of risks, such as cybersecurity and ethical AI use, further complicate the project landscape [11].

2.2 Software Development Life Cycle (SDLC) Models and Their Evolution

The SDLC structures development into requirements, design, implementation, testing, deployment, and maintenance [12]. Over time, several models have been proposed:

- The Waterfall model follows a strict linear sequence, ensuring traceability but offering little flexibility [1].
- The Spiral model (Boehm, 1986) integrates iteration with explicit risk management, making it suitable for complex and high-risk projects [14].
- The V-Model aligns each development stage with corresponding testing activities, strengthening validation but increasing rigidity [1].

- The Agile model abandons linearity, focusing instead on iterative cycles, stakeholder feedback, and incremental delivery [15].

2.3 Agile vs. Waterfall: Comparative Perspectives

The Waterfall model, introduced by Royce in 1970, established a sequential approach to software development. It was widely adopted in industries such as aerospace, defense, and government, where compliance, predictability, and extensive documentation were essential [16]. However, its rigidity often resulted in high costs when requirements evolved during a project.

By contrast, the Agile Manifesto (2001) marked a cultural and methodological shift, promoting adaptability, collaboration, and incremental delivery [17]. Agile practices encourage frequent stakeholder involvement and iterative cycles, making them better suited for dynamic markets.

The differences between the two paradigms can be summarized in Table 2.1.

Table 2.1. Comparison between Waterfall and Agile methodologies

Criterion	Waterfall Model	Agile Methodology
Process Structure	Linear, sequential phases (requirements → deployment).	Iterative, incremental cycles (sprints, continuous delivery).
Flexibility	Low – changes are costly once a phase is complete.	High – requirements can evolve throughout development.
Customer Involvement	Limited – mainly at start and final delivery.	Continuous – active involvement at every iteration.
Documentation	Comprehensive and detailed specifications.	Lightweight – emphasis on working software.
Risk Management	Risks often discovered late in the cycle.	Risks addressed incrementally during iterations.
Delivery	One final release at project end.	Frequent releases of functional increments.
Best Suited For	Stable requirements, compliance-driven projects.	Dynamic environments with evolving customer needs.

Hybrid approaches are increasingly common, combining Waterfall's governance with Agile's flexibility. Industry surveys confirm that more than 70% of organizations employ hybrid practices, particularly in finance, healthcare, and government projects [18].

To illustrate these paradigms visually, Figure 2.4 shows Waterfall's linear flow compared to Agile's cyclical iterations.

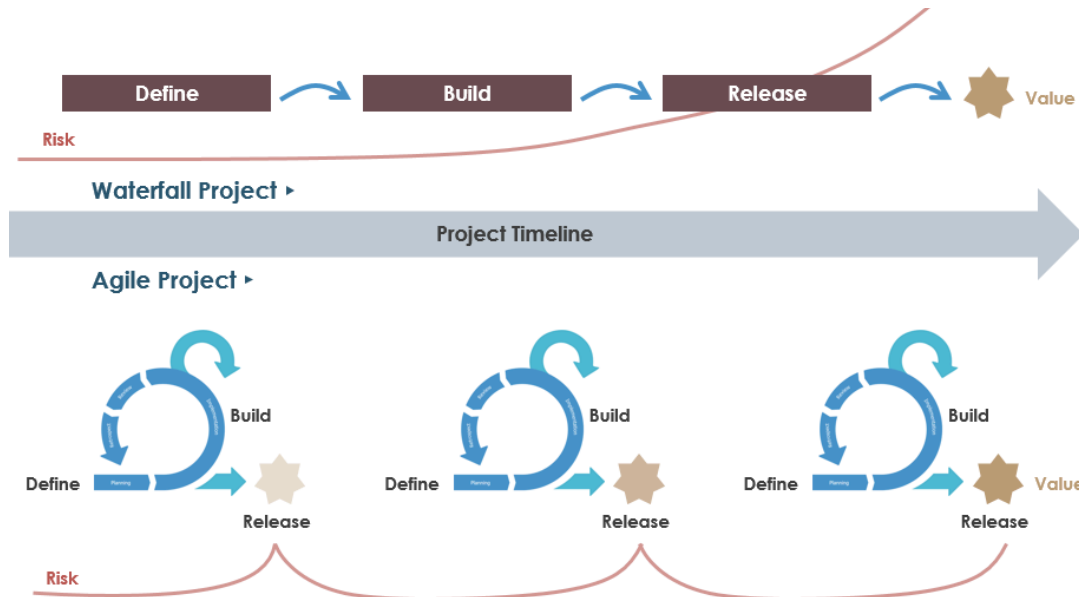


Figure 2.4. Waterfall vs. Agile Development Models

Source: “Agile Product Development vs. Waterfall: Choosing the Right Approach,”
<https://guides.visual-paradigm.com/>

2.4 Agile Principles and Values

The Agile Manifesto articulated four foundational values: individuals and interactions, working software, customer collaboration, and responsiveness to change [17]. These are reinforced by twelve principles that encourage early delivery, sustainable pace, technical excellence, simplicity, and self-organization. Over two decades, these values have proven remarkably durable and have expanded from IT teams to entire organizations. Today, Agile principles inform leadership models, innovation strategies, and enterprise culture [21].

Agile is a family of frameworks with different implementations. Scrum, the most common, structures work into sprints with defined roles and ceremonies to ensure delivery and adaptation. Kanban emphasizes continuous flow through task visualization and WIP limits, favoring

operational contexts [23]. At scale, SAFe integrates Agile, Lean, and DevOps, aligning strategy with execution via program increments, release trains, and portfolio planning [24].

2.5 Challenges in Managing Software Development Projects

Even with mature frameworks, software projects often struggle. Many failures can be traced not to the methodology itself but to unclear distribution of responsibilities, weak alignment between business and technical teams, and lack of ownership for emerging risks. Distributed teams introduce additional communication barriers, while regulatory environments demand accountability in areas like data protection and AI governance.

These persistent challenges highlight the need to examine how roles and responsibilities are structured within Agile teams, a topic addressed in the next chapter.

Chapter 3. Roles and Responsibilities in Agile Teams

Agile methodologies emphasize self-organizing, cross-functional teams, but they leave flexibility in defining how specific roles are distributed. This flexibility is both a strength and a challenge. While it allows organizations to adapt Agile practices to their context, it can also lead to confusion or overlaps in responsibilities. The way in which roles such as Business Analyst (BA), Product Owner (PO), and Project Manager (PM) are allocated has a direct impact on how requirements are discovered, documented, and prioritized [23].

3.1 The Role of the Business Analyst

The **BA** is often the link between stakeholders and development teams. In Agile settings, their contribution goes beyond traditional documentation: they facilitate backlog refinement, ensure clarity in user stories, and define acceptance criteria. When BAs are absent, teams often struggle with ambiguity, resulting in unclear requirements and inconsistent acceptance criteria [23][24].

3.2 The Role of the Product Owner and the Product Manager

The **PO** is accountable for maximizing product value by managing the backlog and prioritizing features. They act as the voice of the customer, translating business needs into backlog items. The **PM**, by contrast, operates at a higher strategic level, shaping product vision, roadmaps, and long-term positioning [25]. In smaller teams, **PO** and **PM** responsibilities may overlap, but in scaled Agile frameworks, separating them is crucial to maintain clarity.

3.3 Comparative Scenarios of Role Distribution

The thesis identifies and evaluates four typical role configurations in Agile teams. These scenarios were tested through practitioner workshops, with results showing how clarity and efficiency differ depending on team composition [28].

Table 3.1. Scenarios of Role Distribution in Agile Teams

Scenario	Strengths	Weaknesses
Only BA	High-quality specifications, structured user stories.	Lack of strategic prioritization; business alignment weaker.
Only PO	Strong business alignment, direct customer focus.	Ambiguity in requirements, weak acceptance criteria.

BA + PO	Balanced: clear user stories + strong prioritization.	Requires collaboration; risk of role overlap.
BA + PO + PM	Clear distribution: BA ensures clarity, PO manages value, PM ensures scope/risk.	Increased coordination costs, potential role conflicts.

The analysis demonstrated that the BA + PO configuration consistently delivered the highest clarity and efficiency in requirements engineering. Adding a PM increased coordination and governance but sometimes introduced role overlap.

3.4 Research Findings

The research confirms that role clarity is essential to Agile success. Teams without a BA faced frequent ambiguity, while teams without a PO lacked strong business direction. The coexistence of BA and PO produced the most effective balance.

In conclusion, Agile projects do not benefit from eliminating roles but from reinterpreting and balancing them according to project context and organizational needs.

Chapter 4. Requirements Engineering and User Stories

This chapter examines how requirements are discovered, analyzed, and expressed in Agile environments, with a particular focus on user stories as the dominant artifact. It highlights the transition from traditional requirements engineering to Agile practices, explores the structure and principles of user stories, and identifies key limitations that motivate the development of an improved standard.

4.1 Sources and Analysis of Requirements

Requirements Engineering (RE) is defined as the systematic process of discovering, documenting, and managing system services and constraints. In traditional, plan-driven approaches, requirements are established upfront as contractual baselines. Agile methodologies challenge this view by treating requirements as evolving, co-created artifacts developed iteratively through collaboration.

Requirements may originate from:

- Stakeholders (users, managers, regulators, sponsors) with divergent goals,
- Business processes, which shape functional workflows,
- Legacy systems, constraining design while revealing expectations,
- Market and competitors, setting benchmarks,
- Regulations, such as GDPR, imposing mandatory compliance rules.

Key methods include interviews, workshops, document analysis, observation, prototyping, and surveys. In Agile, these activities recur throughout the lifecycle, with requirements progressively refined into stories, epics, and acceptance criteria.

Agile introduces issues such as continuously emerging requirements, reliance on unstructured input, unclear role ownership (BA, PO, PM), difficulty in stakeholder alignment, and weaker traceability compared to traditional RE.

The shift is cultural as well as methodological. Requirements move from being static “contracts” to ongoing “conversations.” Analysts must balance ambiguity with clarity, document “just enough,” and maintain continuous communication. This sets the stage for user stories as the preferred Agile artifact.

4.2 The Concept and Structure of User Stories

User stories emerged in the late 1990s within Extreme Programming and were popularized by Jeffries and Cohn as concise alternatives to requirement documents. Their standard structure is:

As a [user], I want [goal] so that [reason].

This captures role, goal, and rationale.

Bill Wake's INVEST model defines effective stories as: Independent, Negotiable, Valuable, Estimable, Small, and Testable. Despite its popularity, teams often struggle with vague wording, incomplete criteria, and oversized stories.

Adaptations include Job Stories (focused on context and motivation), Behavior-Driven Development (BDD) with Given–When–Then criteria, and Story Mapping to visualize journeys and dependencies.

Common issues are ambiguity, oversimplification, time-consuming refinement, and inconsistent adoption across teams. Workshops and surveys confirmed that story writing consumes disproportionate effort, especially when input comes from unstructured sources.

4.3 Epics, Acceptance Criteria, and Definition of Ready

Agile requirements are organized across abstraction levels.

- Epics represent large features split into multiple stories, but often become oversized or outdated.
- Acceptance Criteria define conditions of completion, increasingly using BDD syntax, but are frequently missing or inconsistent.
- Definition of Ready (DoR) serves as a checklist for backlog items before sprint planning but is often applied inconsistently under delivery pressure.

These practices, while useful, suffer from ambiguity, overhead, and inconsistency, highlighting the need for more standardization.

4.4 Limitations of Traditional User Story Writing

User stories, though widely adopted, face recurring limitations:

- Ambiguity from natural language descriptions,
- Inconsistency across teams and organizations,
- Neglect of non-functional requirements such as performance or compliance,
- High overhead for producing INVEST-compliant stories,
- Weak traceability, requiring external tools,
- Difficulty handling complex interactions, technical constraints, or visuals like wireframes.

These limitations underline that the minimalist template is insufficient for large-scale or regulated contexts. Workshops and surveys confirmed the significant time investment and quality variability in story writing.

The chapter concludes that user stories must evolve. The limitations identified motivate the development of the Extended User Story (EUS) framework, which integrates structured acceptance criteria, detailed descriptions, and support for technical and visual elements. This standard will be presented in the following chapter.

Chapter 5. Standardization of User Story Writing

This chapter introduces the Extended User Story (EUS) framework, developed to address limitations of traditional user story formats. It emphasizes the need for standardization in Agile requirements, describes the rules of the EUS format, and presents its validation in practice. The chapter also traces the progression from templates to automation, culminating in systems capable of transforming unstructured discovery inputs into structured backlog items.

5.1 Proposed New Standard for User Story Definition

Studies confirm that poorly defined requirements are the root of most software project failures. Professional experience as a Business Analyst and Product Owner, combined with survey results, revealed that traditional story templates caused both ambiguity and inefficiency. Nearly 20% of the time required to write a user story was wasted on formatting tasks.

The Extended User Story (EUS) format was developed to reduce this waste and provide a standardized, robust approach. It introduces structural and visual rules:

- Delimitation of sections: overview (AS A/I WANT TO/SO THAT), description, acceptance criteria in Gherkin.
- Formatting conventions: keywords in uppercase and bold; buttons in bold purple; error messages in red italics with quotes; informational messages in blue italics with quotes.
- Structured descriptions: tables, bullet points, and references to related stories.
- Acceptance criteria: sequential numbering (AC1, AC2.1, etc.), written in Gherkin syntax, with negatives linked to corresponding positives.

The framework enhances clarity for developers, testers, and stakeholders by unifying business, functional, and technical requirements into a single artifact.

5.2 Validation through Practice and Training

The EUS format evolved through iterative application in professional and educational contexts. In the *Ready for IT* program, over 60 graduates tested the format, with more than 80% later employed as BAs or POs. Many companies adopted EUS internally, reporting reduced refinement times and improved delivery predictability. Employers described EUS as “a new language for requirement specification.”

5.3 Towards Tool-Supported Standardization

While EUS improved clarity, manual enforcement across large backlogs remained effort-intensive. To address this, a Jira extension was developed:

- Predefined sections for overview, description, and acceptance criteria.
- Fields enforcing sequential numbering and Gherkin syntax.
- Built-in formatting for UI elements, error messages, and informational text.
- Workflow integration to flag incomplete or non-compliant stories before sprint entry.

The extension reduced manual effort, supported onboarding, and accelerated backlog preparation in large projects (e.g., e-commerce platforms). It demonstrated the value of deterministic automation—consistency through enforced templates rather than generative creation.

5.4 From Templates to Automated Generation

Despite improvements, manual structuring still consumed resources. The evolution of EUS moved through three layers:

1. Jira extension – embedded deterministic rules into backlog tools.
2. Prompt-driven interface – expanded minimal inputs (AS A/I WANT TO/SO THAT) into fully formatted EUS stories, automatically applying rules.
3. Discovery-to-backlog system – envisioned application capable of ingesting heterogeneous inputs (documents, transcripts, slides) and applying NLP to extract actors, goals, and constraints. These are clustered into epics and decomposed into stories automatically rendered in EUS format.

This layered approach illustrates the path from manual writing → template enforcement → semi-automated generation → AI-driven requirement discovery, with the EUS standard as the backbone ensuring consistency, clarity, and testability across all stages.

Chapter 6. The Jira Extension for Deterministic Automation of User Story Writing

This chapter describes the development and validation of **Specs-Assistant**, a Jira extension created to embed the Extended User Story (EUS) standard directly into Agile workflows. It highlights the motivation, design, functionality, and practical benefits of the extension as a tool for deterministic automation of user story writing.

6.1 Jira as a Platform for Agile Requirements Management

Jira is widely used for Agile backlog and sprint management due to its configurability and integration of workflows, issue types, and custom fields. It provides a central repository for requirements but lacks enforced conventions for user story formatting, often leading to inconsistency and ambiguity. Specs-Assistant addresses this gap by embedding the EUS standard directly into Jira.

6.2 Motivation for Developing the Jira Extension

Manual enforcement of EUS rules requires repetitive formatting and structuring, which consumes nearly 20% of story-writing time. The extension was motivated by two goals:

- Efficiency – reducing time spent on repetitive formatting,
- Consistency – ensuring compliance with EUS rules across all backlog items.

6.3 Design and Functionality of Specs-Assistant

Specs-Assistant integrates into Jira's interface, providing predefined templates and validation features. Its main capabilities include:

- Template creation: auto-insertion of the three EUS sections (overview, description, acceptance criteria).
- Pre-formatted acceptance criteria: automatic numbering (AC1, AC2.1, etc.) with explicit edge and negative cases.
- Formatting enforcement: UI elements, error messages, and system messages pre-styled according to EUS rules.
- Validation support: preventing incomplete or non-compliant stories from being saved.

- **Ease of use:** embedded in Jira's native workflow, minimizing the learning curve.

6.4 User Interface of the Extension

The Specs-Assistant extension was designed with usability in mind, integrating seamlessly into Jira's native workflow. Its interface aims to reduce the cognitive and operational overhead of writing user stories by guiding analysts step by step through a standardized process.

Once installed, the extension activates a main dashboard that serves as the central workspace for Business Analysts and Product Owners. This dashboard provides a structured overview of all epics and their associated user stories, ensuring that requirements are organized hierarchically and consistently. From here, users can:

- Create new backlog items directly with predefined EUS sections (overview, description, acceptance criteria).
- Navigate epics and stories visually, reducing the risk of overlooked dependencies.
- View relationships between epics and their subordinate stories, enabling traceability.

Figure 6.4.3. illustrates this functionality, showing how the extension displays epics at the top level and links each to its respective user stories. This hierarchical view helps maintain alignment between business objectives and detailed requirements.

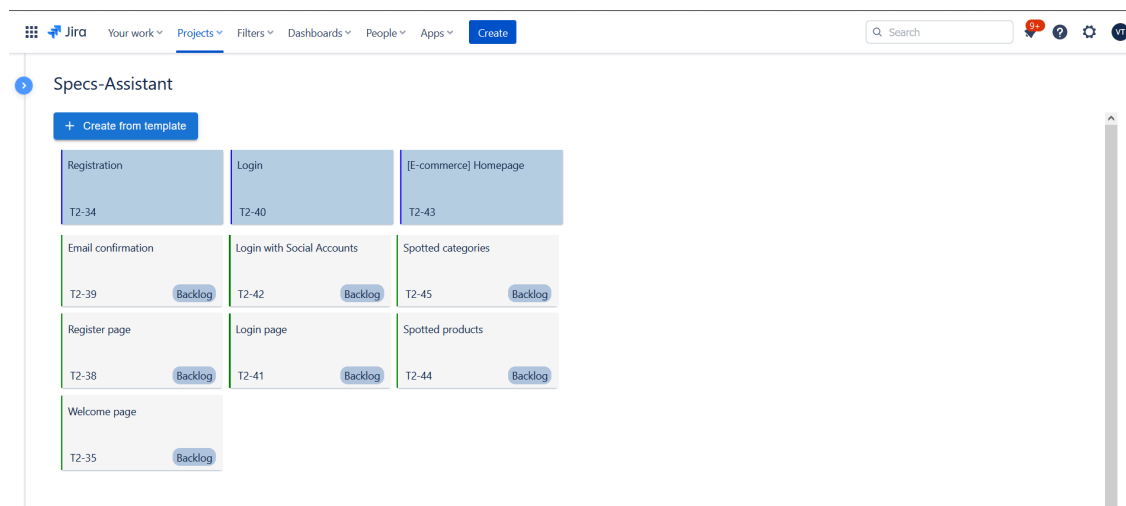


Figure 6.4.3. Dashboard with Epics and Stories

When creating or editing a user story, the interface opens a Jira modal enriched with EUS-specific features:

- All required sections (AS A / I WANT TO / SO THAT, description, acceptance criteria) are pre-populated as placeholders, preventing omissions.
- Custom fields (e.g., regulatory identifiers, traceability tags) can be enforced, ensuring compliance with enterprise governance.
- Formatting rules for UI elements, error messages, and system responses are automatically applied, eliminating repetitive manual work.

The extension also incorporates validation checks that block incomplete or improperly structured stories from being saved. For example, if acceptance criteria are missing or not numbered correctly, the system prompts the user to complete them before proceeding.

This guided experience provides multiple advantages:

- Analysts avoid repetitive formatting tasks.
- Testers and developers benefit from uniform story structures.
- Managers gain more predictable and traceable backlog quality.

By transforming story authoring into a template-driven, validated workflow, Specs-Assistant reduces effort while ensuring that all backlog items meet the Definition of Ready.

6.5 Benefits Observed in Practice

Specs-Assistant delivered clear benefits: a 15–20% reduction in story writing time, consistent adherence to structural standards, faster refinement, and easier onboarding for new team members. By embedding EUS templates directly into Jira, it operationalized standardization and marked the first step toward broader automation in requirements engineering, evolving from deterministic enforcement to intelligent generation and discovery-to-backlog transformation.

Chapter 7 – AI-powered User Story Generation Interface

This chapter explores the transition from deterministic, rule-based approaches to AI-powered automation in user story generation. It highlights the application of OpenAI's GPT-3 API for requirements engineering, emphasizing efficiency, scalability, and adaptability in Agile practices.

The motivation derives from the shortcomings of manual authoring: user stories are often inconsistent, ambiguous, and time-consuming to write. While deterministic methods, such as a Jira extension tailored for e-commerce projects, showed encouraging results (up to 60% reduction in writing effort), their reliance on rigid templates limited adaptability across domains [5]. Large Language Models (LLMs), and particularly GPT-3, offer a more flexible and scalable solution by transforming unstructured inputs into structured requirements aligned with the Extended User Story (EUS) standard.

7.1 Related Work

Several platforms and studies have advanced the automation of user story generation:

- Atlassian Jira has introduced AI functionalities such as recommending formats, auto-generating acceptance criteria, and detecting inconsistencies.
- AgileGen (Zhang et al. [29]) focuses on Gherkin-based testable requirements to ensure alignment between specifications and code, though it addresses later stages of the lifecycle compared to this study.
- Generative AI systems such as ChatGPT and Claude have been increasingly applied to produce draft stories and refine them, offering efficiency gains and safer, context-sensitive outputs.

Recent research emphasizes the lack of large, open datasets for training robust models and highlights the importance of prompt engineering as a strategy for obtaining consistent outputs from AI models [32]. Broader studies on AI in Agile project management underline benefits such as improved backlog management, collaboration, and risk mitigation [5].

7.2 From Manual Templates to AI-Enhanced Generation

The first automation effort relied on a deterministic Jira extension. It embedded predefined templates composed of:

1. Persona, user objective, and motivation.

2. Detailed functionality descriptions.
3. Acceptance criteria in Gherkin format.

While efficient within e-commerce, this method lacked generalizability. Templates had to be recreated manually for each domain, limiting scalability [5].

The rule-based approach was domain-specific. Different industries (healthcare, finance, retail) require distinct terminology, personas, and compliance factors. This exposed the limits of deterministic automation and motivated the shift toward **AI-driven generation** [5].

By integrating OpenAI's GPT-3 API, the research introduced automation that could scale across domains. With prompt engineering, GPT-3 produced EUS-compliant stories from minimal input, ensuring clarity and adaptability.

- Figure 7.2.1. shows how Business Analysts provide only user type and intended action, and the assistant generates a complete story.

The screenshot displays the 'Create your user story' interface. On the left, there is a form with two input fields. The first field, labeled 'AS A', contains the text 'visitor'. Below it, 'examples:' are listed as 'visitor', 'registered user', and 'customer'. The second field, labeled 'I WANT TO', contains the text 'access a login page'. Below it, 'examples:' are listed as 'access a login page' and 'send an order'. A purple 'Generate' button is positioned below the form. On the right, the generated user story is shown in a light gray box. It starts with 'AS A visitor', followed by 'I WANT TO access a login page' and 'SO THAT I can log in to the website and access personalized content or services.' Below this, the 'USER ACCEPTANCE CRITERIA' section lists three criteria: 'AC1: Title: Visitor can access the login page' with 'GIVEN: The visitor is on the website homepage', 'WHEN: The visitor clicks on the "Login" button or navigates to the login page URL', and 'THEN: The login page is displayed'; 'AC2: Title: Visitor cannot access the login page if already logged in' with 'GIVEN: The visitor is already logged in', 'WHEN: The visitor clicks on the "Login" button or navigates to the login page URL', and 'THEN: The visitor is redirected to their account page or dashboard'; and 'AC3: Title: Visitor cannot access the login page with invalid credentials' with 'GIVEN: The visitor is on the login page', 'WHEN: The visitor enters invalid login credentials (e.g., incorrect username or password)', and 'THEN: An error message is displayed, stating that the login was unsuccessful'. A purple 'Copy' button is located at the bottom right of the generated story box.

Figure 7.2.1. BA-Assistant client interface

- Figure 7.2.2. illustrates the architecture: client interface, application server, embedding models, and GPT engine interact to process inputs and generate responses.

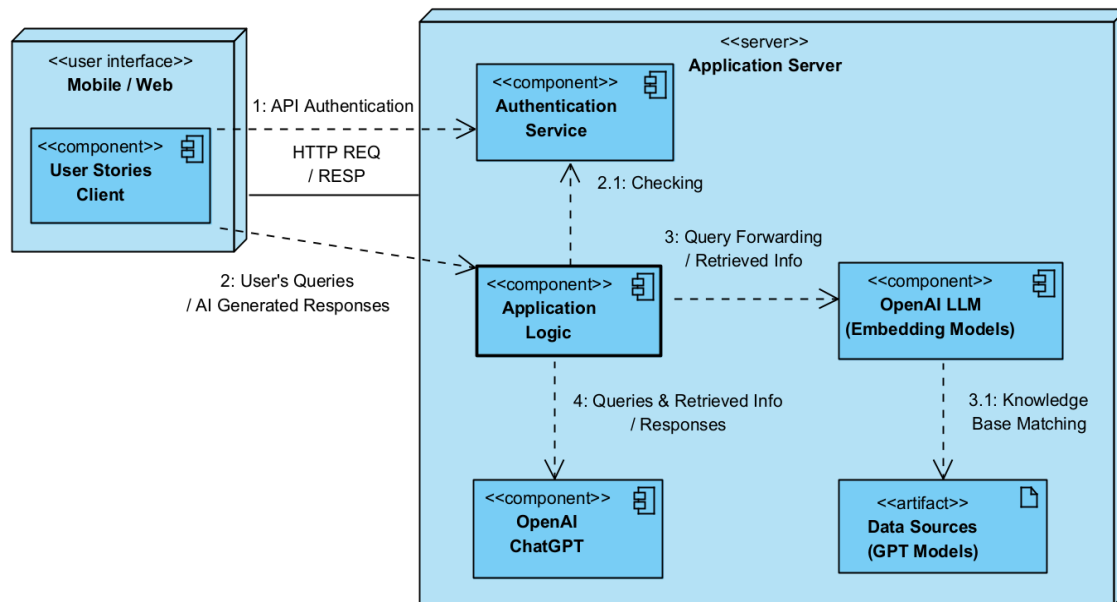


Figure 7.2.2. High-Level Architecture of the Initial AI-Driven Prototype

Prompt engineering aligned AI outputs with Agile practices by using structured examples and iterative testing, as shown in Figure 7.2.3.

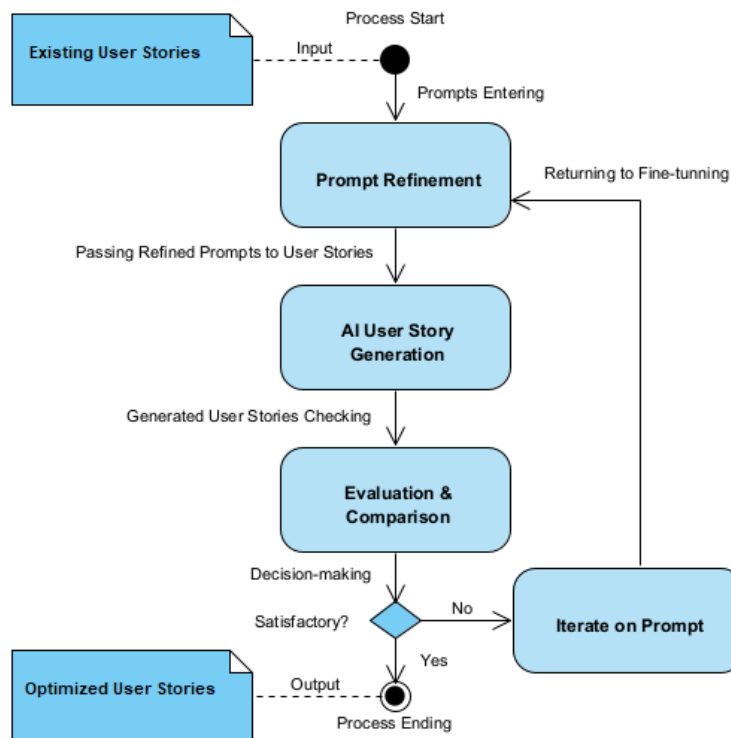


Figure 7.2.3. Prompt Engineering Workflow

7.3 Comparative Evaluation of AI Models

Two models were tested: ChatGPT and Gemini, using equivalent prompts.

Table 7.2. Comparative Evaluation of AI Models (ChatGPT vs. Gemini).

Model	Strengths	Weaknesses
ChatGPT	Strong contextual understanding; detailed acceptance criteria; higher alignment with business value	Occasionally verbose
Gemini	Concise, clear, well-structured outputs	Less detailed; sometimes omits acceptance criteria

Both produced coherent, EUS-compliant stories, but ChatGPT demonstrated superior contextual depth for complex cases [5].

7.4 Findings and Challenges

AI-powered story generation brings significant improvements to requirements engineering by reducing manual effort, ensuring scalability across industries, maintaining consistency with the Extended User Story (EUS) standard, and incorporating learning capabilities that allow models to evolve through user feedback.

At the same time, several challenges must be addressed, including safeguarding sensitive requirements to ensure data privacy, providing explainability so that stakeholders can understand and trust AI outputs, and preserving human oversight to maintain contextual accuracy and stakeholder alignment.

The chapter concludes that AI shifts user story definition from repetitive manual tasks to a scalable, adaptable process, providing stronger support for Agile teams and enterprise-scale projects.

Chapter 8. AI-Driven Multimodal Requirement Discovery

The chapter introduces Discovery AI, an AI-powered platform that tackles ambiguity, miscommunication, and weak traceability in early Agile discovery by embracing the multimodal, conversational nature of real meetings. Instead of treating requirements as purely textual artifacts, Discovery AI ingests audio recordings, notes, and visual cues and—through ASR, RAG, and LLMs such as GPT-4o, fine-tuned Llama 3.2, and Gemini 2-Flash—outputs structured Agile requirements (epics, user stories, themes). Developed iteratively, the first complete version (as published in recent academic work) shows measurable improvements in requirement clarity/traceability, categorization accuracy, and documentation speed, while a later streamlined variant is discussed in subsequent sections [26]. By positioning meetings as dynamic knowledge sources and combining real with synthetic data, Discovery AI directly addresses prior assumptions that requirements are strictly textual or deterministic and demonstrates cross-industry adaptability [27].

8.1 Foundational Technologies in AI-Driven Requirement Discovery

ASR (Automatic Speech Recognition). Discovery AI’s modular ASR layer converts meeting audio into time-aligned, speaker-attributed transcripts (noise reduction, speaker diarization, timestamping/segmentation), enabling robust downstream extraction even with accents, overlap, or informal dialogue.

RAG (Retrieval-Augmented Generation). RAG composes answers by retrieving semantically relevant context from a vector database and injecting it into prompts before generation. In Discovery AI it is used during refinement, user-story generation, and categorization, reducing hallucination and ensuring continuity across similar projects via memory-augmented generation.

LLMs. Discovery AI combines: GPT-4o for first-pass understanding/drafting, fine-tuned Llama 3.2 for structural rigor and Agile alignment, and Gemini 2-Flash for fast summarization. Zero/few-shot behavior, domain generalization (fintech, healthtech, e-commerce, logistics), and prompt templates (managed in Langfuse) provide controllability.

Supporting infrastructure. Qdrant enables fast semantic search across meetings, past stories, literature, and synthetic datasets; Langfuse versions prompts, A/B tests instructions, and tracks performance for reliability and traceability. Model selection balances precision on requirement tasks, summarization speed/cost, and compatibility with RAG/fine-tuning; modularity lets teams swap components as needed. [26]

8.2 From Fragmented Tools to Integrated Discovery Pipelines

Where toolchains are typically fragmented (separate transcription, summarization, story mapping, documentation), Discovery AI integrates the pipeline from raw multimodal input (audio/video/notes) to structured Agile documentation (epics and user stories) without requiring pre-structuring or user expertise in underlying AI.

8.3 Dual-Phase Generation Approach

Discovery AI separates **drafting** and **refinement** to combine speed and rigor.

GPT-4o ingests ASR transcripts (segmented/diarized), Gemini 2-Flash summaries, and RAG-retrieved knowledge to produce broad EUS-shaped drafts of epics and user stories—fast, creative, domain-general, and suitable for surfacing candidate requirements.

Llama 3.2 enforces the Agile template (“As a..., I want..., so that...”), adds domain-specific terminology, resolves redundancy/ambiguity, and harmonizes stories across epics, yielding production-ready artifacts aligned with retrieved context.

Table 8.3. GPT-4o drafting and Llama 3.2 refinement

Feature	GPT-4o Drafting (Phase 1)	Llama 3.2 Refinement (Phase 2)
Speed	Very high	Moderate (deeper analysis)
Context breadth	Broad input coverage	Domain-specific focus
Agile compliance	Partial (needs validation)	Full template compliance
Hallucination risk	Higher without RAG	Lower (validation via refinement)
Output quality	Coarse, exploratory	Precise, production-ready

This two-stage design ensures outputs are grounded, efficiently generated, Agile-aligned, and refined for coherence and traceability.

8.4 System Overview

Discovery AI integrates knowledge construction, meeting understanding, and LLM refinement:

- Knowledge base construction. Expert literature, organizational docs, meeting transcripts, and synthetic data are embedded and indexed in Qdrant, powering RAG and grounding outputs in best practices and project realities.
- Meeting understanding & product discovery. Audio/video are transcribed (ASR), summarized (Gemini 2-Flash), and semantically analyzed (GPT-4o) with prompt templates versioned in Langfuse; synthetic transcripts augment scarce or edge scenarios.
- Fine-tuning a specialized LLM. Llama 3.2 is fine-tuned on curated, versioned datasets (via Langfuse) and deployed through Hopsworks Model Registry as a cloud API, enabling seamless integration with tools like Jira, Notion, or ClickUp.

Discovery AI's knowledge base grounds outputs in verified, domain-relevant information, reducing hallucinations and ensuring accurate user stories. Unlike tools limited to transcripts, it integrates diverse sources for semantic retrieval, aligning live requirements with best practices and project history.

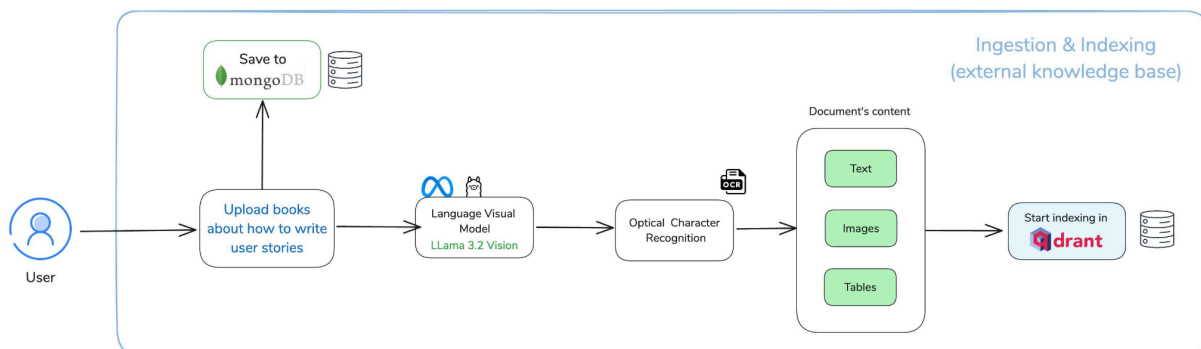


Figure 8.4.1. External Knowledge base Workflow in Discovery AI

Business meetings are rich but unstructured sources of requirements. The Meeting Understanding module of Discovery AI converts raw audio/video recordings into structured, retrievable insights. Recordings are uploaded to Amazon S3, ensuring durability and reprocessing capability. They are transcribed by Whisper ASR, chosen for robustness in multilingual, noisy, and overlapping speech contexts. Transcripts, enriched with timestamps and speaker diarization, are stored in MongoDB for traceability.

Next, transcripts are processed by GPT-o1 (GPT-4o), which extracts latent meaning and candidate requirements, guided by prompt templates versioned in Langfuse. A parallel summarization pathway with Gemini 2-Flash produces concise meeting overviews, enabling

rapid indexing and quick insights. Both detailed transcripts and summaries are indexed in Qdrant, supporting semantic retrieval.

In addition, the architecture includes a synthetic data generation module, where GPT-o1/4o creates simulated meeting transcripts that cover edge cases, rare requirements, and underrepresented domains. These synthetic discussions enrich the knowledge base, addressing dataset scarcity in requirement engineering.

Together, Qdrant (semantic similarity search) and MongoDB (structured metadata) form a dual-layer storage strategy: Qdrant enables concept-driven retrieval, while MongoDB ensures traceability. Through this hybrid pipeline, illustrated in Figure 8.4.2, Discovery AI transforms unstructured meetings into structured Agile artifacts, grounded in both real and synthetic data.

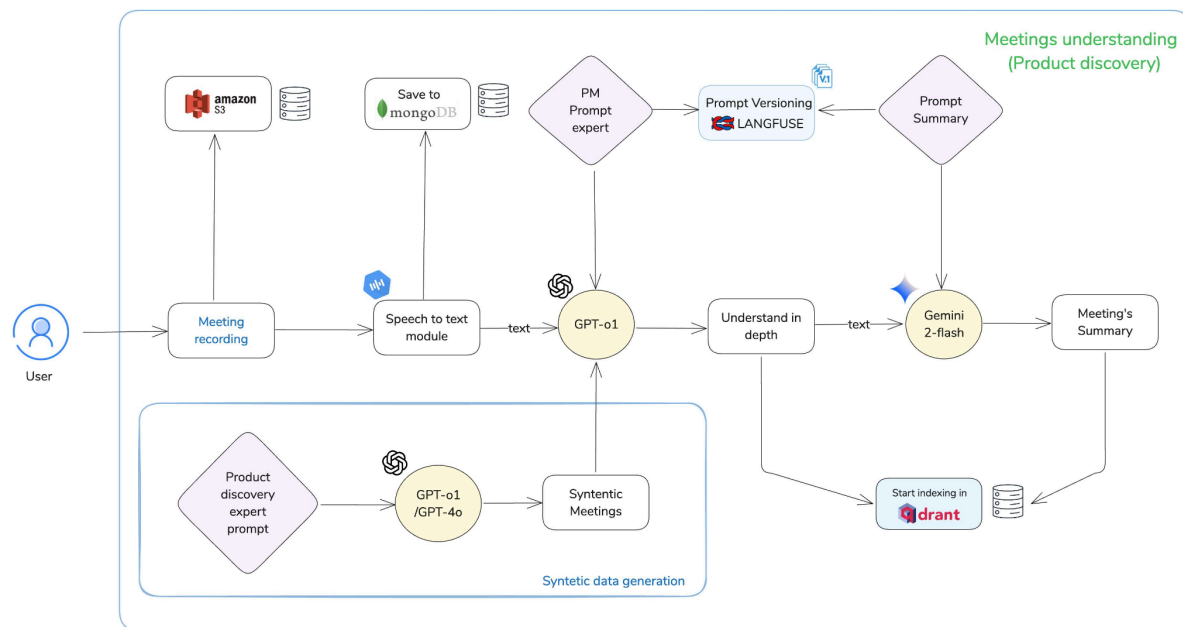


Figure 8.4.2. Meeting Understanding and Product Discovery Pipeline — showing audio ingestion, Whisper transcription, GPT-4o insight extraction, Gemini summarization, and storage in MongoDB and Qdrant.

The diagram in Figure 8.4.3. illustrates the workflow employed in Discovery AI to fine-tune a large language model (LLM) for the specific task of generating Agile user stories. While general-purpose models (e.g., GPT-4o) can provide broad semantic coverage, their outputs often lack the structural rigor and contextual precision required in Agile requirement engineering. To address this limitation, Discovery AI incorporates a dedicated fine-tuning pipeline based on

Llama 3.2, supported by systematic dataset preparation, versioning, and deployment mechanisms.

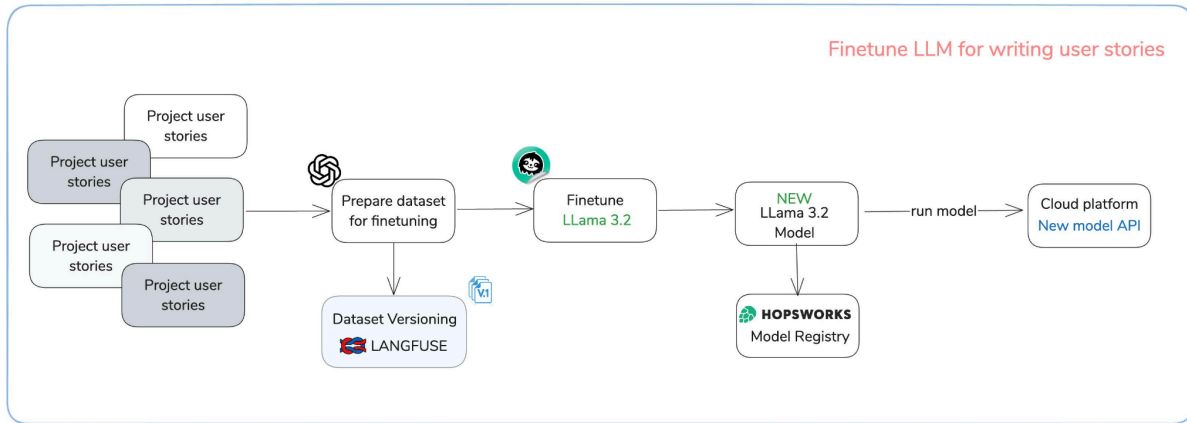


Figure 8.4.3. Fine-Tuning Workflow for Writing User Stories in Discovery AI

8.5 Inference Workflow

The Inference Workflow represents the operational phase of Discovery AI, where the system is invoked by end users (e.g., product managers, product owners or business analysts) to generate epics and user stories for a given project. Unlike the training and fine-tuning pipelines, which are largely offline processes, the inference workflow operates in near real time and must balance efficiency, accuracy, and contextual grounding. The process is illustrated in Figure 8.5, and it unfolds in four sequential stages:

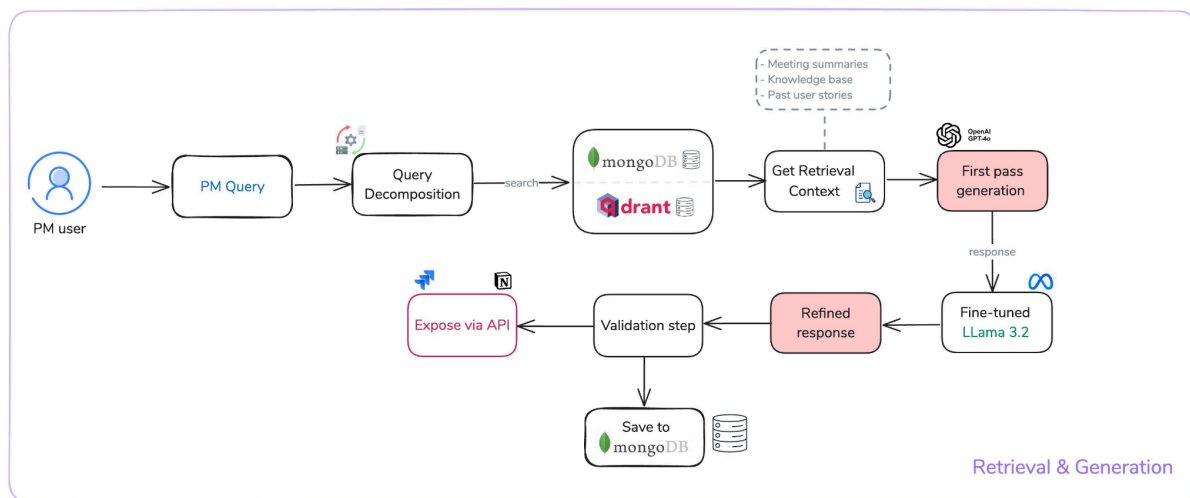


Figure 8.5. Inference Workflow in Discovery AI

Chapter 9: A Simplified Iteration of Discovery AI with GPT-5

This chapter introduces the second iteration of Discovery AI, redesigned to take advantage of the multimodal and reasoning capabilities of GPT-5. The first version relied on a complex, multi-model pipeline involving Whisper for transcription, GPT-4o for first-draft generation, Gemini 2-Flash for summarization, and Llama 3.2 for refinement. While this approach was functional, it introduced significant latency, high maintenance overhead, and considerable integration complexity.

With the release of GPT-5, which integrates speech-to-text, summarization, reasoning, and Agile-compliant generation within a single model, a simplified pipeline became possible. As shown in Figure 9.1, the new system consolidates functionalities that were previously distributed across several components, reducing dependencies and improving robustness.

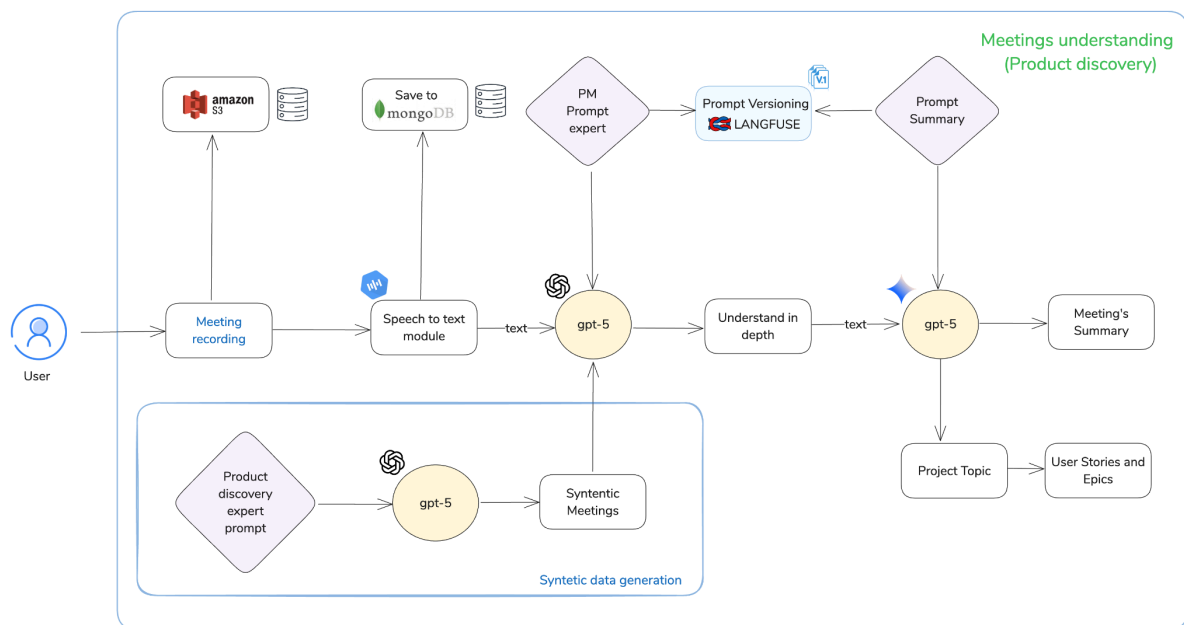


Figure 9.1. Simplified Discovery AI with GPT-5 – from meeting recordings to user stories and epics in a streamlined workflow.

9.1 System Design

The simplified pipeline begins with meeting notes and recordings, stored in Amazon S3 for durability and reproducibility. Processed transcripts, summaries, and Agile artifacts are preserved in MongoDB, ensuring auditability. GPT-5 handles transcription natively, eliminating the need for Whisper, and simultaneously performs semantic enrichment, capturing implicit requirements during transcription. It further integrates reasoning, summarization, and structured generation,

producing user stories and epics directly in canonical Agile format, complete with acceptance criteria.

Prompt management remains essential, though its scope is reduced. Using Langfuse, prompts are versioned and optimized to ensure reproducibility and consistent outputs. Instead of orchestrating across multiple models, Langfuse now interacts primarily with GPT-5, focusing on structured extraction, topic classification, and Agile story generation.

Three types of structured prompts are central to ensuring consistency and usability of outputs. The first prompt enforces a JSON schema for extracting key meeting insights such as discussion points, requirements, risks, and decisions, making them directly machine-readable. The second organizes discussions into percentage-based topic classifications, quantifying how much of a meeting was devoted to predefined categories like online store, booking, or scheduling. The third defines the format for generating epics and user stories, requiring Agile-compliant structures with acceptance criteria written in Gherkin syntax, so that the outputs can be integrated immediately into project management workflows.

9.2 Comparative Analysis: Old vs. New Iteration

The simplification achieved through GPT-5 is best illustrated by a comparison between the two iterations of Discovery AI. Table 9.1 summarizes the main differences.

Table 9.1: Comparison between Discovery AI v1 (multi-model pipeline) and v2 (simplified with GPT-5)

Feature / Dimension	Discovery AI v1 (Multi-Model)	Discovery AI v2 (GPT-5 Simplified)
Speech-to-Text	Whisper ASR (separate module)	Native GPT-5 multimodal processing
Summarization	Gemini 2-Flash	Integrated summarization within GPT-5
First Draft Generation	GPT-4o	GPT-5
Refinement	Fine-tuned Llama 3.2	GPT-5 (no separate refinement needed)

Knowledge Retrieval (RAG)	Qdrant + MongoDB	Qdrant + MongoDB (reduced overhead)
Synthetic Data Generation	GPT-4o guided by prompts	GPT-5 directly, more integrated
Average Latency per Project	2–3 minutes	< 30 seconds (estimated)
Maintenance Complexity	High (multiple models and pipelines)	Low (single foundation model)

This comparison highlights the trade-off between fine-grained modularity (v1) and integrated simplicity (v2).

9.3 Interface and Usability

Discovery AI v2 also introduces a redesigned interface (Figure 9.3), which integrates all stages of the workflow into a single dashboard. Users can upload meeting files, inspect structured insights, review topic classifications, and access AI-generated user stories with acceptance criteria. These outputs can be edited and exported directly into project management platforms such as Jira or ClickUp. This accessibility reflects Agile values of transparency and collaboration, allowing both technical and non-technical stakeholders to engage with AI-generated requirements.

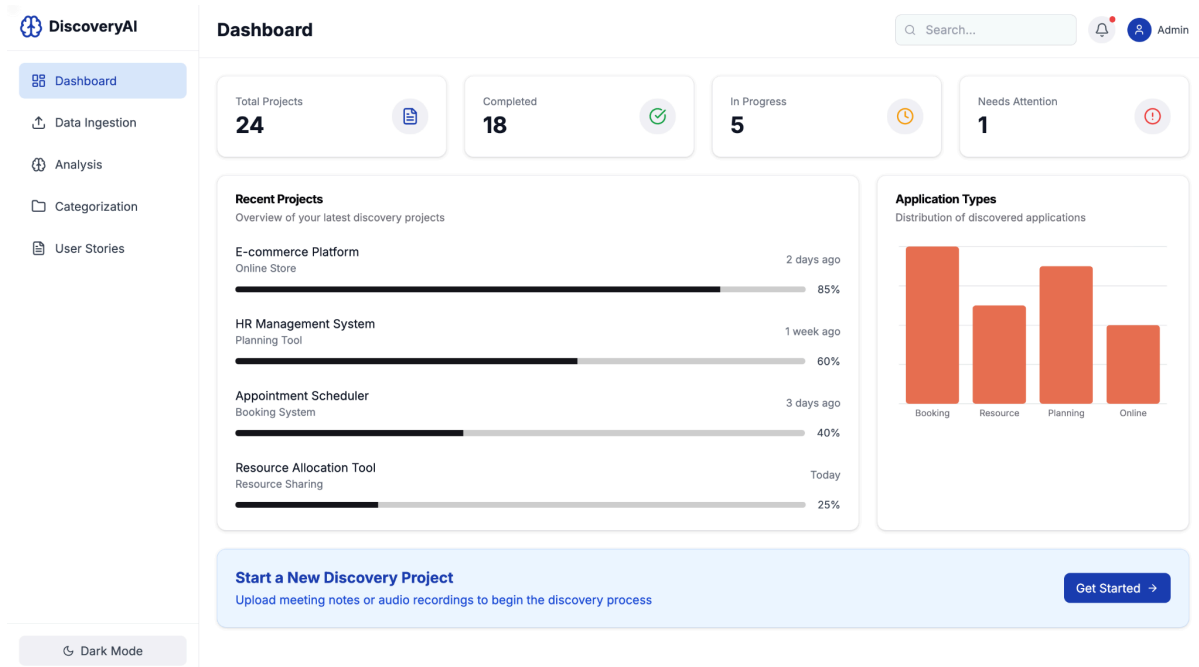


Figure 9.3. Discovery AI interface
Dashboard overview

The revised interface in Discovery AI v2 addresses these shortcomings by enabling direct interaction with all stages of the pipeline through a single dashboard (see Figure 9.4.).

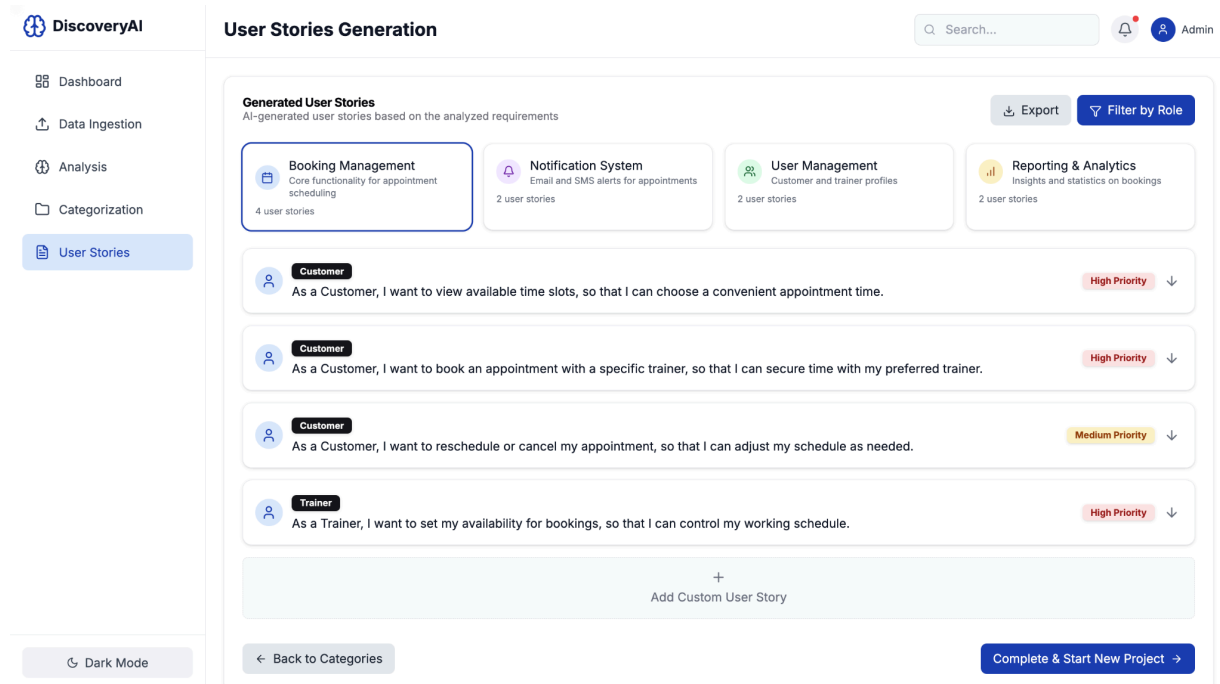


Figure 9.4. Discovery AI interface - Generated epics and user stories with acceptance criteria

9.4 Opportunities and Limitations

While the simplified pipeline offers clear efficiency and usability gains, it also introduces new dependencies. The system is now fully reliant on GPT-5; any limitations or biases of the model affect all stages of the workflow. Furthermore, the removal of a separate refinement stage reduces opportunities for domain-specific customization. Future research may explore hybrid strategies that retain the efficiency of GPT-5 while introducing lightweight fine-tuned components for specialized contexts.

Chapter 10. Conclusions

10.1. Concluzii generale

This research tackled the critical challenge of requirements definition by introducing standardized, semi-automated, and AI-enhanced solutions to improve clarity, traceability, and efficiency. Building on evidence that unclear requirements drive over 80% of project failures, it proposed the Extended User Story (EUS) standard and evolved from a deterministic Jira extension to an AI-driven, multimodal prototype capable of generating user stories from transcripts, recordings, and documents. Experiments confirmed the feasibility, scalability, and practical value of these tools in both corporate and educational contexts.

10.2. Original Contributions

The doctoral research brought multiple original contributions, both theoretical and practical:

- Theoretical and methodological contributions:
 - Definition of the Extended User Story (EUS) format, a new user story structure that combines an overview, detailed description, and acceptance criteria in Gherkin format, together with visual markers that enhance clarity and usability.
 - Introduction of formatting rules that differentiate interface elements, system messages, and error messages to reduce ambiguity.
- Technical contributions:
 - Development of a JIRA extension that integrates the EUS structure into Agile workflows and accelerates the writing and validation of user stories in real projects.
 - Creation of an AI-based prototype that integrates OpenAI's API to generate user stories based on user prompts or meeting notes, achieving up to 60% time reduction in the story definition phase.
 - Proposal of a to be published AI architecture capable of extracting requirements, structured through an orchestrated pipeline.
- Scientific and educational contributions:
 - Publication of two peer-reviewed articles that describe the proposed architecture and the transition from deterministic to AI-driven requirement engineering.

- Integration and validation of the EUS format in the Ready for IT training program, where more than 80% of graduates successfully secured roles as Business Analysts or Product Owners, several of whom implemented the standard within their hiring companies.

10.3. Future Research Directions

Future research should focus on integrating tools like Jira and Confluence for full automation, fine-tuning GPT models on datasets such as NeoDataset for better domain adaptation, and expanding Discovery AI to process spoken interactions. Additional directions include developing agent-based validation for consistency and compliance, implementing explainability and safety modules for regulated industries, and testing the system across domains like healthcare, fintech, retail, and public services to ensure robustness.

10.4. Work Synthesis

The doctoral research pursued a sustained focus on requirement discovery, automation, and AI integration in Agile software engineering, combining theoretical innovation with practical validation in industry and academia. Core contributions include two ISI-indexed articles: *“Transforming user story definition: From deterministic to AI-powered automation”* and *“Enhancing agile requirement discovery with AI-driven multimodal systems and synthetic user story generation”*, which underpin Chapters 7 and 8.

Further outputs include BDI publications on AI microservices, Scrum team selection, and the Discovery AI platform, presented at Smart Cities conferences. Professionally, the work extended into training over 80 specialists through the Ready for IT Academy, achieving strong placement rates, and developing educational content later incorporated into Chapters 5 and 6. The author also delivered academic lectures, organized the “Life in Tech” event, and actively engaged in major international conferences, ensuring continuous feedback from practitioners. Recognition in Forbes 30 Under 30 Romania (2023) as co-founder of Genezio further highlights the intersection of research and entrepreneurial impact.

Overall, the thesis establishes a methodological and technological foundation for advancing AI-driven software requirement discovery, validated across corporate, academic, and interdisciplinary contexts.

References

- [1] I. Sommerville, *Software Engineering*, 10th ed. Pearson, 2015.
- [2] Standish Group, *CHAOS Report*, 2020.
- [3] M. Cohn, *User Stories Applied: For Agile Software Development*. Addison-Wesley, 2004.
- [4] G. Lucassen, F. Dalpiaz, J. M. E. van der Werf, and S. Brinkkemper, “The Use and Effectiveness of User Stories in Practice,” *Requirements Engineering*, vol. 21, no. 3, pp. 383–403, 2016.
- [5] **A.-P. Avasiloaie**, A. Semenescu, E.-C. Popovici, “Transforming User Story Definition: From Deterministic to AI-Powered Automation,” *Romanian Journal of Information Technology and Automatic Control*, vol. 35, no. 2, pp. 59–72, 2025. WOS:001522929000005
- [6] B. Ramesh and L. Cao, “Agile Requirements Engineering Practices and Challenges,” *IEEE Software*, vol. 27, no. 6, pp. 60–67, 2010.
- [7] A. Ferrari, P. Spoletini, and S. Gnesi, “Ambiguity and Tacit Knowledge in Requirements Elicitation Interviews,” *Requirements Engineering Journal*, vol. 21, no. 3, pp. 333–355, 2016.
- [8] H. Hassani, X. Huang, and E. Silva, “Artificial Intelligence in Requirements Engineering: Opportunities and Challenges,” *Information Systems*, vol. 108, p. 102012, 2022.
- [9] N. Bennett and G. Lemoine, “What VUCA Really Means for You,” *Harvard Business Review*, 2014.
- [10] S. Kraus, T. Clauss, M. Breier, J. Gast, A. Zardini, and V. Tiberius, “The Economics of COVID-19: Initial Empirical Evidence on How Family Firms in Five European Countries Cope with the Corona Crisis,” *International Journal of Entrepreneurial Behavior & Research*, vol. 27, no. 2, 2021.
- [11] H. Kerzner, *Project Management: A Systems Approach to Planning, Scheduling, and Controlling*, 13th ed. Wiley, 2022.
- [12] R. S. Pressman and B. R. Maxim, *Software Engineering: A Practitioner’s Approach*, 9th ed. McGraw-Hill, 2019.
- [13] B. Boehm, “A Spiral Model of Software Development and Enhancement,” *ACM SIGSOFT Software Engineering Notes*, vol. 11, no. 4, pp. 14–24, 1986.
- [14] K. Schwaber and J. Sutherland, *The Scrum Guide*, 2020.
- [15] W. W. Royce, “Managing the Development of Large Software Systems,” in *Proc. IEEE WESCON*, 1970.
- [16] K. Beck et al., *Manifesto for Agile Software Development*, 2001. [Online]. Available: <https://agilemanifesto.org/>
- [17] PMI & Digital.ai, *Agile Adoption Report*, 2024.
- [18] J. Humble and D. Farley, *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Addison-Wesley, 2010.

-
- [19] L. Leite, A. Rocha, C. Silva, T. Conte, and C. R. B. de Souza, "Large Language Models in Agile Practices: Opportunities and Risks," *Empirical Software Engineering*, vol. 29, 2024.
- [20] S. Denning, *The Age of Agile: How Smart Companies Are Transforming the Way Work Gets Done*. Amacom, 2018.
- [21] D. Anderson, *Kanban: Successful Evolutionary Change for Your Technology Business*. Blue Hole Press, 2010.
- [22] Scaled Agile Inc., *SAFe Framework*, Version 6.0, 2023.
- [23] H. Sharp and H. Robinson, "Some Social Factors of Software Engineering: The Maturing of Agile Methods," *Information and Software Technology*, vol. 48, no. 6, pp. 541–552, 2006.
- [24] B. Ramesh, L. Cao, and R. Baskerville, "Agile Requirements Engineering Practices and Challenges: An Empirical Study," *Information Systems Journal*, vol. 20, no. 5, pp. 449–480, 2010.
- [25] R. Pichler, *Agile Product Management with Scrum*. Addison-Wesley, 2010.
- [26] **A.-P. Avasiloaie**, A. Semenescu, E.-C. Popovici, and I.-C. Chiva, "Enhancing agile requirement discovery with AI-driven multimodal systems and synthetic user story generation," *Scientific Bulletin Series C: Electrical Engineering and Computer Science*, ISSN 2286 - 3540
- [27] **A.-P. Avasiloaie**, A. Semenescu, E.-C. Popovici, R. Craciunescu, I.-C. Chiva, "Leveraging NVIDIA AI technologies in the development of REQAPP: a machine learning platform for gathering and defining requirements for smart cities applications," 2025. "Smart Cities International Conference (SCIC) Proceedings," V12, (Sep. 2025), pp. 465–480.
- [28] **A.-P. Avasiloaie**, A. Semenescu, E.-C. Popovici, R. Craciunescu, I.-C. Chiva, "AI-driven decision support for SCRUM team selection in smart city software development projects," 2025. "Smart Cities International Conference (SCIC) Proceedings," V12, (Sep. 2025), pp. 453–464.
- [29] I. Altawaiha and A. Al-Hgaish, "ClassDiagGen Tool: Fine-Tuning the GPT-3 Model for Automated Class Diagram Generation from Textual Descriptions," *Research Square*, 2024. [Online]. Available: <https://www.researchsquare.com/> [Accessed: Aug. 12, 2025].
- [30] O. Buruk, "Academic Writing with GPT-3.5: Reflections on Practices, Efficacy and Transparency," *arXiv preprint*, 2023. doi: 10.48550/arXiv.2304.
- [31] T. -C. Ureche, E.-C. Popovici, S. Halunga, **A.-P. Avasiloaie**, L. Boicescu, D. Țurcanu, "Architecting Secure Smart Infrastructures with AI Microservices and Autonomous Agents: A State-of-the-Art Review and Healthcare Use Case," 2025. "IEEE International Black Sea Conference on Communications and Networking," 23–26 June 2025. Chisinau, Moldova
- [32] T.-C. Ureche, E.-C. Popovici, S. Halunga, **A.-P. Avasiloaie**, L. Boicescu, D. Țurcanu, "Architecting Secure Smart Infrastructures with AI Microservices and Autonomous Agents: A State-of-the-Art Review and Healthcare Use Case," 2025. "IEEE International Black Sea Conference on Communications and Networking," 23–26 June 2025, Chisinau, Moldova.
- [33] A. Haile, *AI-Driven Software Testing Automation: Machine Learning Strategies for Performance Optimization in Distributed Networks*, 2024.

- [34] A. Haile, Innovating Software Testing with AI and Machine Learning: Automation for Efficient Distributed Network Management, 2024.
- [35] S. Mansour, “Atlassian welcomes AI to the team,” Atlassian Blog, 2023. [Online]. Available: <https://www.atlassian.com/blog/announcements/atlassian-intelligence-ga> [Accessed: Aug. 12, 2025].
- [36] T. Pîrcălabu, N. Țăpuș, and A. I. Damian, “Programming assistance based on Neural AI OS Platform,” Revista Română de Informatică și Automatică, p. 43, 2024. doi: 10.33436/v34i4y202404.
- [37] S. M. T. H. Rimon, Leveraging Artificial Intelligence in Business Analytics For Informed Strategic Decision-Making: Enhancing Operational Efficiency, Market Insights, And Competitive Advantage, 2024.
- [38] M. Stephen, The Intersection of Technology and Writing Support: How ChatGPT is Changing Writing Center Dynamics, 2024.
- [39] S. Zhang et al., “Empowering Agile-Based Generative Software Development through Human-AI Teamwork,” arXiv preprint, 2024. doi: 10.48550/arXiv.2407.15568.