Methods for integrating large language models into requirements management in agile methodologies

Oleh Zaiats[1,*,†], Dmytro Mykhalyk[1,†], Vasyl Yatsyshyn[1,†], Oleh Pastukh[1,†], Dmytro Uhryn[2,†]

[1] *Ternopil Ivan Puluj National Technical University, Ruska, 56, Ternopil, 46001, Ukraine*
[2] *Yuriy Fedkovych Chernivtsi National University, Kotsiubynskoho, 2, Chernivtsi, 58002, Ukraine*

Abstract
Agile software development methodologies rely on effective requirements management to maintain alignment between evolving business needs and delivered functionality. This article investigates the integration of Large Language Models (LLMs) into five widely used Agile frameworks: Scrum, Kanban, Extreme Programming (XP), the Scaled Agile Framework (SAFe), and Lean Software Development with a focus on enhancing requirement capture, refinement, prioritization, traceability, and feedback processing. For each methodology, the study maps specific LLM capabilities to native workflow stages, illustrating opportunities for automation and augmentation without undermining core Agile principles. The findings demonstrate that LLMs can reduce ambiguity, accelerate refinement cycles, and strengthen stakeholder–developer collaboration, while also highlighting risks such as hallucinations, bias, and data privacy concerns. The article concludes that effective integration requires tailored strategies, human oversight, and alignment with organizational priorities, and identifies future research directions including domain-specific fine-tuning and long-term impact evaluation.

Keywords:
Large Language Models, Agile, requirements engineering, Scrum, Kanban, SAFe, XP, Lean

1. Introduction

Effective requirements management is one of the most critical success factors in software development [1]. In Agile environments, where change is expected and feedback loops are short, requirements must be captured, refined, prioritized, and validated in a way that preserves clarity while enabling rapid delivery. However, Agile teams often face recurring challenges: requirements arrive in inconsistent formats, stakeholder input may be incomplete or ambiguous, and the pace of change can lead to gaps in traceability or overlooked dependencies. These challenges are particularly pronounced in complex software systems where quality management and systematic evaluation approaches are essential [3,19].

Recent advances in Large Language Models (LLMs), such as GPT-4, Claude, and other transformer-based systems, have created new opportunities to address these challenges. Trained on vast amounts of text data, LLMs excel at understanding, classifying, summarizing, and generating natural language. These capabilities position them as promising assistants in the requirements engineering process – from transforming raw stakeholder input into structured backlog items, to maintaining requirement quality across the entire development lifecycle. The integration of artificial intelligence and data science approaches in software development [2] demonstrates the potential for automated assistance in traditionally manual processes.

[1]*ITTAP'2025: 5th International Workshop on Information Technologies: Theoretical and Applied Problems, October 22-24, 2025, Ternopil, Ukraine, Opole, Poland*
* Corresponding author.
† These authors contributed equally.
ozaiats@tntu.edu.ua (O. Zaiats); myhalyk_d@tntu.edu.ua (D. Mykhalyk); yacyshyn@tntu.edu.ua (V. Yatsyshyn); oleg.pastuh@gmail.com (O. Pastukh); d.ugryn@chnu.edu.ua (D. Uhryn);
0009-0002-3386-5943 (O. Zaiats); 0000-0001-9032-695X (D. Mykhalyk) 0000-0002-5517-6359 (V. Yatsyshyn); 0000-0002-0080-7053 (O. Pastukh); 0000-0003-4858-4511 (D. Uhryn);

This article explores methods for integrating LLMs into requirements management processes within five Agile methodologies: Scrum, Kanban, Extreme Programming (XP), the Scaled Agile Framework (SAFe), and Lean Software Development. For each methodology, we map LLM capabilities to native workflow stages, propose concrete enhancement points, and illustrate the changes using process diagrams. The goal is to identify integration strategies that deliver measurable value – such as reduced ambiguity, faster backlog preparation, improved prioritization, and better traceability – while respecting the core principles of each methodology and established software quality frameworks [4].

While the potential of LLMs in Agile requirements management is significant, their adoption requires careful consideration of risks, including hallucinated outputs, embedded bias, over-reliance on automation, and data confidentiality concerns. As such, this study emphasizes human-in-the-loop approaches, where LLMs augment but do not replace human judgment in quality-critical decisions [3].

The findings contribute both to the theoretical understanding of AI-assisted requirements engineering and to its practical application in Agile settings. They provide a foundation for future research into domain-specific fine-tuning, integration into project management tools, and empirical studies on productivity and quality outcomes.

2. Overview of Agile Methodologies and Approaches to Requirements Management

Agile software development encompasses a variety of frameworks, each with its own philosophy, workflow structure, and approach to managing requirements. Although the Agile Manifesto emphasizes “working software over comprehensive documentation” and “responding to change over following a plan,” requirements remain central to ensuring product success. Understanding how requirements are gathered, refined, and maintained across different Agile methods is a prerequisite for identifying effective Large Language Model (LLM) integration points.

2.1. Scrum

Scrum is an iterative framework structured around fixed-length sprints, typically lasting two to four weeks [5,6]. In figure 1, the main stages of Scrum and relationships between them are displayed.

Requirements are captured primarily in the Product Backlog as user stories, often accompanied by acceptance criteria. The process of backlog refinement (also known as grooming) ensures that items are well-defined before sprint planning. Stakeholder feedback during Sprint Reviews allows for continuous adjustment of requirements.

Scrum pipeline stages:

Stage 1 – Capture high-level requirements from stakeholders

Stage 2 – Convert into user stories

Stage 3 – Add user stories to Product Backlog

Stage 4 – Refine backlog items with team input

Stage 5 – Select backlog items for the sprint

Stage 6 – Sprint execution

Stage 7 – Sprint review

Stage 8 – Validate and update requirements. Proceed to stage 3.

Scrum’s requirements management is structured yet flexible, balancing a central backlog with iterative refinement. Its defined events and artifacts promote clarity and adaptability throughout the development cycle.

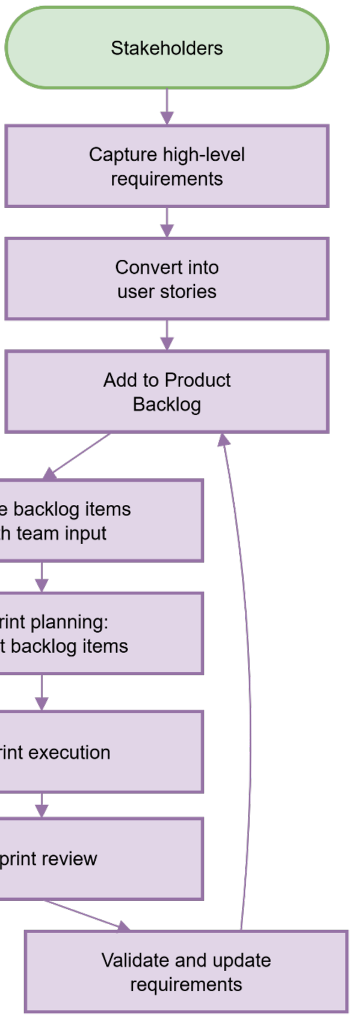

Stakeholders
Capture high-level requirements
Convert into user stories
Add to Product Backlog
Refine backlog items with team input
Sprint planning: select backlog items
Sprint execution
Sprint review
Validate and update requirements

Figure 1. Scrum pipeline

2.2. Kanban

Kanban focuses on continuous flow rather than fixed iterations. Requirements enter the workflow as work items on the Kanban board and progress through stages such as “To Do,” “In Progress,” and “Done.” Kanban minimizes formal documentation, relying on visual management and limiting work in progress [7,8]. The pipeline stages of Kanban are shown in Figure 2.

Kanban Pipeline includes:

Stage 1 – Gather raw requirements.

Stage 2 - Add structured requirement as Kanban card.

Stage 3 - Prioritize according to business needs.

Stage 4 - To Do.

Stage 5 - In Progress.

Stage 6 - Done.

Stage 7 - Continuous or batch release.

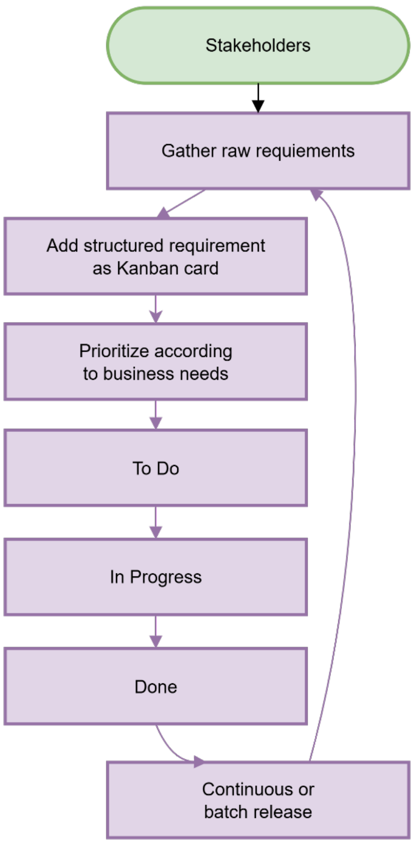

Stakeholders
Gather raw requiements
Add structured requirement as Kanban card
Prioritize according to business needs
To Do
In Progress
Done
Continuous or batch release

Figure 2. Kanban pipeline

Kanban's visual and continuous approach allows for rapid adaptation to change but relies heavily on the quality of requirement intake and ongoing communication to ensure clarity.

2.3. Extreme Programming (XP)

XP emphasizes technical excellence, continuous integration, and frequent releases. Requirements are often represented as story cards discussed directly with an on-site customer. This method integrates requirement specification tightly with automated testing, using practices such as test-driven development (TDD) [9].

The stages of the Extreme Programming pipeline are shown in Figure 3.

Extreme Programming Pipeline includes following stages:

Stage 1 - Planning Game, where customer and team collaborate to define user stories and release planning.

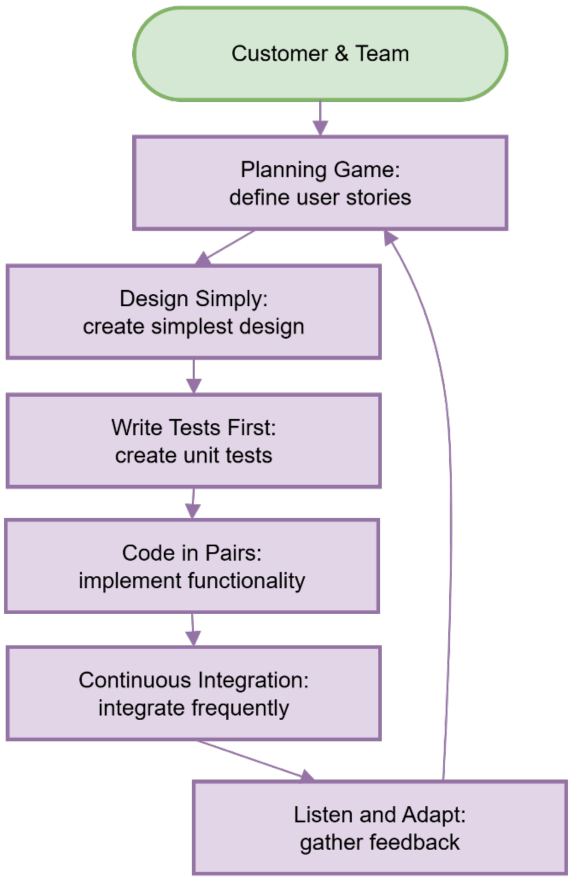
Stage 2 - Design Simply: Create the simplest design that works, using metaphors and CRC cards.

Stage 3 - Write Tests First: Create unit tests before coding.

Stage 4 - Code in Pairs: Implement code with pair programming.

Stage 5 - Continuous Integration: Integrate and test frequently.

Stage 6 - Listen and Adapt: Gather feedback from running software and adjust stories accordingly.



Customer & Team
Planning Game: define user stories
Design Simply: create simplest design
Write Tests First: create unit tests
Code in Pairs: implement functionality
Continuous Integration: integrate frequently
Listen and Adapt: gather feedback

Figure 3. Extreme Programming (XP) pipeline

XP's lightweight, test-first approach to requirements promotes rapid responsiveness to customer needs while maintaining high-quality standards through continuous testing.

2.4. Scaled Agile Framework (SAFe)

SAFe (Scaled Agile Framework) is the world's leading framework for implementing Agile practices at enterprise scale, created by Dean Leffingwell in 2011 to address the challenge of coordinating multiple Agile teams across large organizations while maintaining the flexibility and responsiveness of traditional Agile methods [10,11]. The framework features a multi-level structure spanning Portfolio, Large Solution, Program, and Team levels, with Program Increments (PIs) providing 8-12 week planning cycles that synchronize multiple teams, Agile Release Trains (ARTs) organizing collections of 5-12 Agile teams working together, hierarchical requirements flowing from Portfolio Epics through Capabilities and Features down to Stories, and continuous delivery mechanisms with built-in feedback loops. SAFe combines Lean, Agile, and DevOps principles into a comprehensive system that helps organizations deliver value faster while maintaining quality and alignment across all levels of the enterprise.

SAFe Pipeline stages are shown in figure 4.

Stages of SAFe are:

Stage 1 – Portfolio Strategy - define portfolio epics and strategic themes at the portfolio level.

Stage 2 – Solution Planning - break down portfolio epics into capabilities and program epics.

Stage 3 – Program Planning - decompose capabilities/epics into features during PI planning.

Stage 4 – Team Execution - break features into stories and execute in iterations within Program Increments.

Stage 5 – Continuous Delivery - deliver value incrementally across ARTs with regular inspect and adapt cycles.

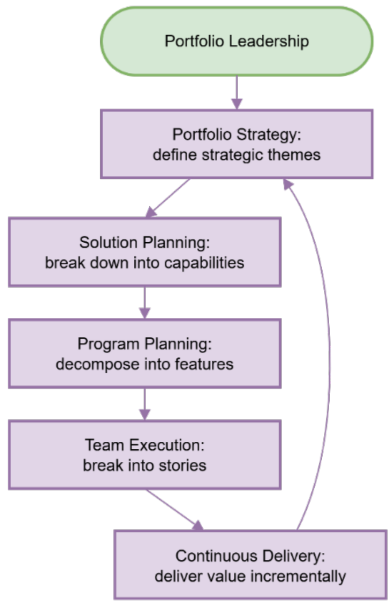

Portfolio Leadership
Portfolio Strategy: define strategic themes
Solution Planning: break down into capabilities
Program Planning: decompose into features
Team Execution: break into stories
Continuous Delivery: deliver value incrementally

Figure 4. Scaled Agile Framework (SAFe) pipeline

SAFe provides a structured yet flexible approach to scaling Agile beyond individual teams. By organizing work hierarchically from strategic portfolio epics down to implementable user stories, and coordinating delivery through Program Increments and Agile Release Trains, SAFe enables large organizations to maintain Agile's benefits – rapid response to change, continuous improvement, and customer focus – while providing the coordination and governance needed at enterprise scale.

The framework's emphasis on alignment, transparency, and continuous delivery makes it particularly valuable for organizations building complex products that require coordination across multiple teams, departments, and even business units. With over 20,000 organizations worldwide using SAFe, it has proven its effectiveness in helping enterprises achieve business agility in today's rapidly changing market conditions.

2.5. Lean Software Development

Lean emphasizes eliminating waste and delivering maximum value to the customer as quickly as possible. Requirements are kept minimal and often validated by quick prototyping. Decision-making is decentralized, and documentation is limited to essential artifacts [12]. Figure 5 shows the pipeline of Lean software development.

Pipeline stages:

Stage 1 - Define what customers truly value and eliminate unnecessary work.

Stage 2 - Visualize workflow from concept to delivery, identifying waste and bottlenecks.

Stage 3 - Establish continuous workflow by removing delays and optimizing process steps.

Stage 4 - Implement demand-driven delivery where work starts only when customers need it.

Stage 5 - Continuously improve processes through regular feedback and waste elimination.

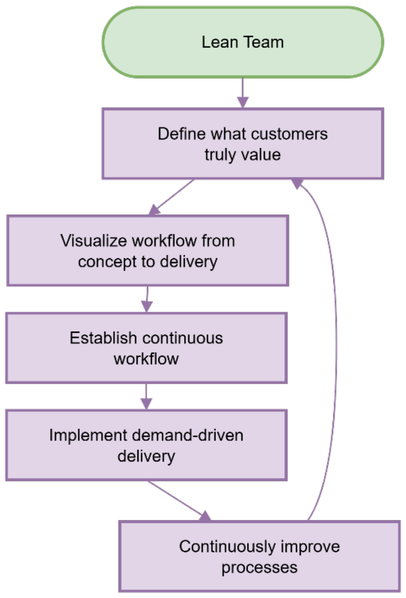

Lean Team
Define what customers truly value
Visualize workflow from concept to delivery
Establish continuous workflow
Implement demand-driven delivery
Continuously improve processes

Figure 5. Lean Software Development pipeline

Lean’s streamlined approach enables fast delivery and adaptability but depends on clear prioritization and disciplined validation to avoid pursuing low-value work.

Comparative analysis of requirement management characteristics in Agile frameworks are displayed in table 1.

Table 1
Comparative Table of Requirement Management Characteristics in Agile Frameworks

Criterion	Scrum	Kanban	XP	SAFe	Lean
Formality of Requirements	Medium – user stories + acceptance criteria	Low – lightweight cards	Medium – story cards + tests	High – epics, features, stories	Low – minimal documentation
Change Frequency	Moderate to high per sprint	Continuous	Continuous	Controlled at PI boundaries	Continuous
Stakeholder Involvement	High during sprint review	High via ongoing feedback	Very high (on-site customer)	High at program and team levels	High via direct user validation
Traceability	Basic – backlog links	Minimal	Test-driven traceability	Strong hierarchical traceability	Minimal
Tooling Support	Jira, Azure DevOps, Trello	Jira, Trello	Jira, custom boards	Jira, Rally, VersionOne	Kanban boards, custom tools

3. Types of Large Language Models and Their Potential in Requirements Management

Large Language Models (LLMs) are advanced AI systems trained on extensive corpora of text to perform a wide range of language-related tasks, including comprehension, summarization, classification, and generation of natural language. Their transformer-based architectures enable them to capture contextual relationships in text with high accuracy, making them a potentially transformative tool in requirements engineering (RE) for Agile environments. This section outlines the classification of LLMs, their capabilities relevant to requirements management, and the risks associated with their application.

3.1. Classification of LLMs

LLMs can be categorized along several dimensions that determine their suitability for Agile requirements workflows [13,14]

1. By scope of training
    - General-purpose models – trained on diverse datasets from multiple domains (e.g., OpenAI GPT-4, Anthropic Claude, Google Gemini). They are adaptable to various contexts but may require prompt engineering for domain-specific accuracy.
    - Domain-specific models – fine-tuned on specialized corpora, such as requirements documentation, industry standards, or regulatory guidelines. These models (e.g., legal-specific LLMs, medical domain LLMs) provide higher accuracy in their respective domains. Research has demonstrated the effectiveness of automated processing and analysis of domain-specific texts, such as medical documentation, using machine learning and deep learning approaches [18], which parallels the potential for specialized requirements engineering models.
2. By accessibility
    - Proprietary models – commercial offerings with API-based access and high performance but limited transparency in training data (e.g., GPT-4, Claude, Gemini).
    - Open-source models – available for self-hosting and customization (e.g., LLaMA 2, Falcon, MPT), offering greater control over data privacy and compliance.
3. By deployment model
    - Cloud-hosted LLMs – operated by the provider, enabling scalable use but raising concerns over confidentiality and compliance.
    - On-premises/self-hosted LLMs – deployed within organizational infrastructure, ensuring data control but requiring more computational resources and maintenance.

Understanding the classification of LLMs is crucial for selecting an appropriate model that aligns with the organization's domain, security requirements, and technical capabilities.

3.2. Capabilities Relevant to Agile Requirements Management

LLMs can enhance different stages of the requirements engineering process in Agile contexts, from initial backlog creation to continuous refinement and traceability. The following capabilities are the most relevant for practical integration[16,17]:

1. Automated requirements classification
    - Sorting and tagging requirements into categories such as functional, non-functional, constraints, and quality attributes.
    - Detecting duplicates or contradictions that may otherwise be overlooked during manual backlog review, particularly in large, multi-team environments.
2. Transformation of user stories into technical specifications

   - Expanding a short, business-oriented user story into a full set of acceptance criteria.
   - Suggesting relevant architectural considerations or component-level implications, which can be reviewed by technical leads before development begins.
   - In some cases, even generating draft interface definitions or pseudo-code that illustrate the intended functionality.
3. Requirements traceability
   - Maintaining bidirectional links between requirements, design elements, test cases, and deployment tasks.
   - Automatically updating traceability matrices when backlog items are modified, merged, or split, helping teams meet compliance or audit requirements without significant manual overhead.
4. Validation and completeness checks
   - Detecting vague or ambiguous terms (e.g., "fast," "user-friendly") and prompting clarification.
   - Highlighting missing elements such as acceptance tests, performance thresholds, or security considerations.
   - Comparing related backlog items to ensure consistent terminology and avoid subtle conflicts.
5. Integration with project management tools
   - Embedding AI assistance directly in platforms like Jira, Azure DevOps, or Trello, enabling in-context refinement of requirements.
   - Automating prioritization based on changing business metrics, dependencies, or risk assessments.
   - Providing analytics on backlog quality over time, such as the ratio of well-formed to incomplete user stories.

LLMs bring multi-faceted benefits to Agile requirements management – not only by accelerating repetitive tasks like classification and traceability but also by improving requirement clarity, consistency, and adaptability to changing business priorities.

3.3. Risks and Limitations

While the promise of LLMs in Agile requirements management is considerable, their adoption comes with notable caveats that teams must address before large-scale deployment [15]:

- Hallucinations and factual inaccuracies – LLMs occasionally produce content that sounds plausible but is factually wrong or entirely fabricated. This is particularly dangerous in requirements engineering, where such errors could propagate through design and testing before being detected.
- Data privacy and confidentiality – Cloud-hosted LLMs often require sending prompts and context outside the organization's network. For projects involving sensitive or regulated information, this raises compliance risks and may require either anonymization of inputs or on-premises deployment.
- Bias in generated content – Pre-trained models can inadvertently introduce bias in requirement phrasing, prioritization, or suggested solutions. While often subtle, such bias may influence product decisions in ways that conflict with inclusivity or fairness goals.
- Over-reliance on automation – There is a danger that teams start trusting AI-generated backlog items, acceptance criteria, or dependencies without sufficient human review. Agile thrives on collaboration, and over-delegating decisions to an algorithm may reduce team engagement and shared ownership.

- Operational considerations – Frequent API calls to commercial LLMs can significantly increase project costs, while self-hosting large models may require specialized hardware, higher energy consumption, and ongoing model maintenance.

To harness the benefits of LLMs without undermining Agile principles, organizations must combine AI assistance with robust governance – including human-in-the-loop review, data handling policies, and continuous monitoring of output quality. Large Language Models can be classified by their scope, accessibility, and deployment approach, each offering distinct trade-offs between flexibility, control, and resource requirements. Their capabilities extend well beyond simple text generation – supporting classification, refinement, traceability, validation, and tool integration – making them highly relevant to Agile requirements management.

At the same time, effective adoption demands awareness of potential risks: inaccuracies, bias, data security concerns, over-reliance on automation, and operational costs. These challenges highlight the importance of tailored integration strategies that account for the unique workflows of different Agile methodologies. The following section builds on these insights by exploring how LLMs can be systematically embedded into Scrum, Kanban, XP, SAFe, and Lean pipelines, including practical examples and visual process models.

4. Methods and Potential Integration Paths of LLMs into Agile Methodologies

The application of LLMs in Agile environments should be adapted to the specific workflow and artifacts of each methodology. This section proposes practical integration strategies for five widely used frameworks – Scrum, Kanban, Extreme Programming (XP), SAFe, and Lean – and illustrates their use through diagrams and a detailed case study for Scrum.

4.1. LLM-Enhanced Scrum Pipeline

Scrum's iterative structure and well-defined artifacts make it a good candidate for introducing AI-assisted automation. The integration points for LLMs can be positioned to support backlog refinement, sprint planning, test case generation, and post-sprint feedback analysis.

LLM-enhanced Scrum pipeline:

Stage 1.1 – Capture high-level requirements from stakeholders

Stage 1.2 (LLM). At this stage LLMs can be used to normalize raw notes (deduplicate, cluster by theme, extract actors/actions/constraints, flag ambiguities). Auto-draft clarifying questions and a short glossary per feature; optional translation for multilingual input. Then LLM can draft user stories based on this input.

Stage 2.1 – Convert into user stories – now this stage is more about validating LLM-created user-stories, making adjustments if necessary before adding user stories to the backlog

Stage 2.2 (LLM) - LLM can Transform cleaned notes into user stories that follow INVEST criteria; generate initial acceptance criteria (e.g., Gherkin) and highlight missing NFRs. Propose epic/feature groupings and link possible duplicates before adding to the backlog.

Stage 3 – Add user stories to Product Backlog

Stage 4.1 – Refine backlog items with team input

Stage 4.2 (LLM) - LLM can suggest story splitting/merging, dependency hints, risk and impact notes, surface contradictions across stories. Provide estimate ranges as "hints" only and mark under-specified stories for PO review.

Stage 5.1 – Select backlog items for the sprint

Stage 5.2 (LLM) - LLM can recommend priority based on simple heuristics (e.g., value, risk, dependencies, due dates) and check team capacity assumptions. Warn about prerequisite gaps or stories that lack clear acceptance criteria.

Stage 6 – Sprint execution

Stage 7.1 – Sprint review

Stage 7.2 (LLM) - LLM can summarize stakeholder feedback, map it to new/updated stories, and note terminology drift. Produce a short change log and traceability suggestions back to epics/tests.

Stage 7.3 – Validate and update requirements based on Product Owner input and using LLM summary. Continue with stage 3.

Figure 6 presents a visualization of the modified Scrum pipeline with integrated LLM models.

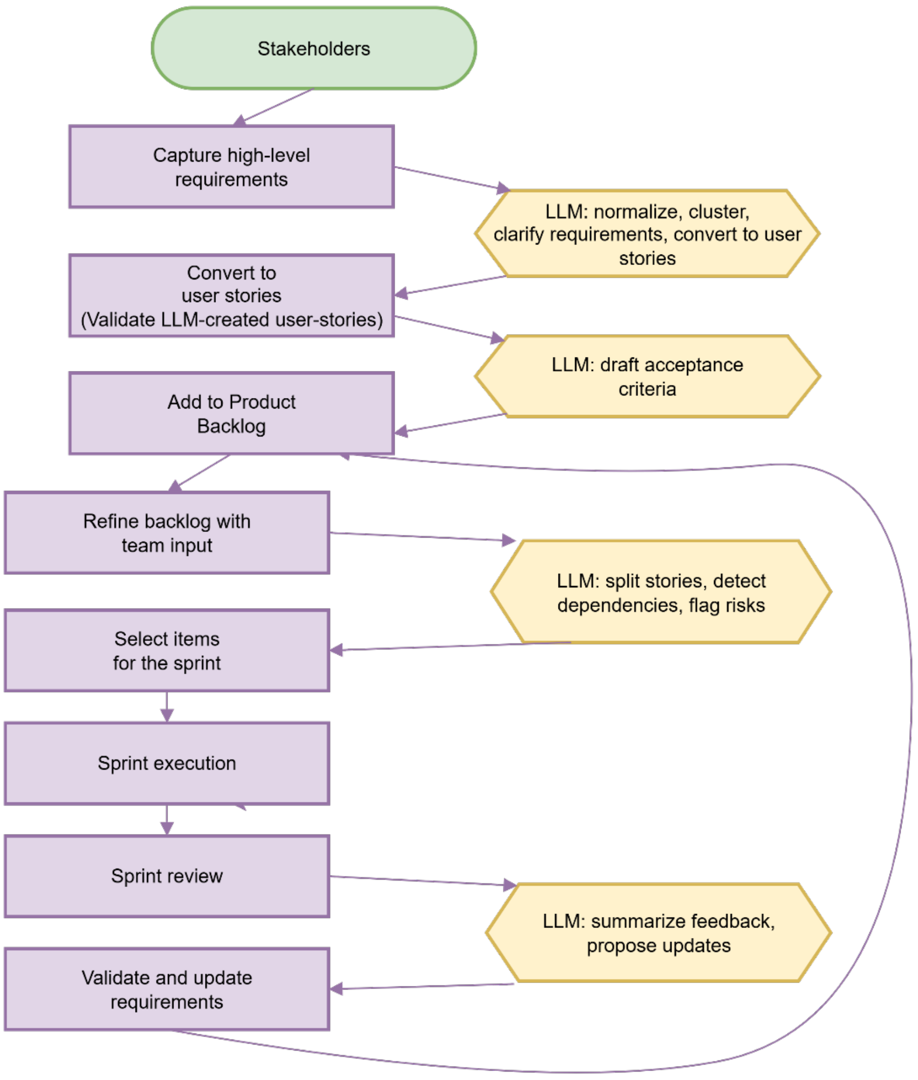

Stakeholders
Capture high-level requirements
LLM: normalize, cluster, clarify requirements, convert to user stories
Convert to user stories (Validate LLM-created user-stories)
LLM: draft acceptance criteria
Add to Product Backlog
Refine backlog with team input
LLM: split stories, detect dependencies, flag risks
Select items for the sprint
Sprint execution
Sprint review
LLM: summarize feedback, propose updates
Validate and update requirements

Figure 6. LLM-Enhanced Scrum Pipeline

LLMs can improve Scrum requirements management by clarifying stakeholder inputs, converting them into well-structured user stories with acceptance criteria, and assisting in backlog refinement through dependency detection, risk highlighting, and story adjustments. They support sprint planning with prioritization hints and capacity checks, aid development by suggesting acceptance tests and edge cases, and post-sprint by analyzing feedback to update the backlog. This reduces ambiguity, speeds refinement, and strengthens traceability while keeping human oversight central.

4.2. LLM-enhanced Kanban pipeline

While Kanban excels at transparency and adaptability, its minimal documentation approach can lead to vague or incomplete cards if the intake process is not well structured. Large Language Models can improve Kanban's requirement handling by transforming unstructured stakeholder inputs into well-formed cards, enriching them with acceptance criteria, and maintaining clarity as work progresses. They can also assist in prioritization, detect bottlenecks, and generate meaningful release summaries without breaking Kanban's lightweight workflow. Figure 7 displays the modified Kanban pipeline with LLMs models integration.

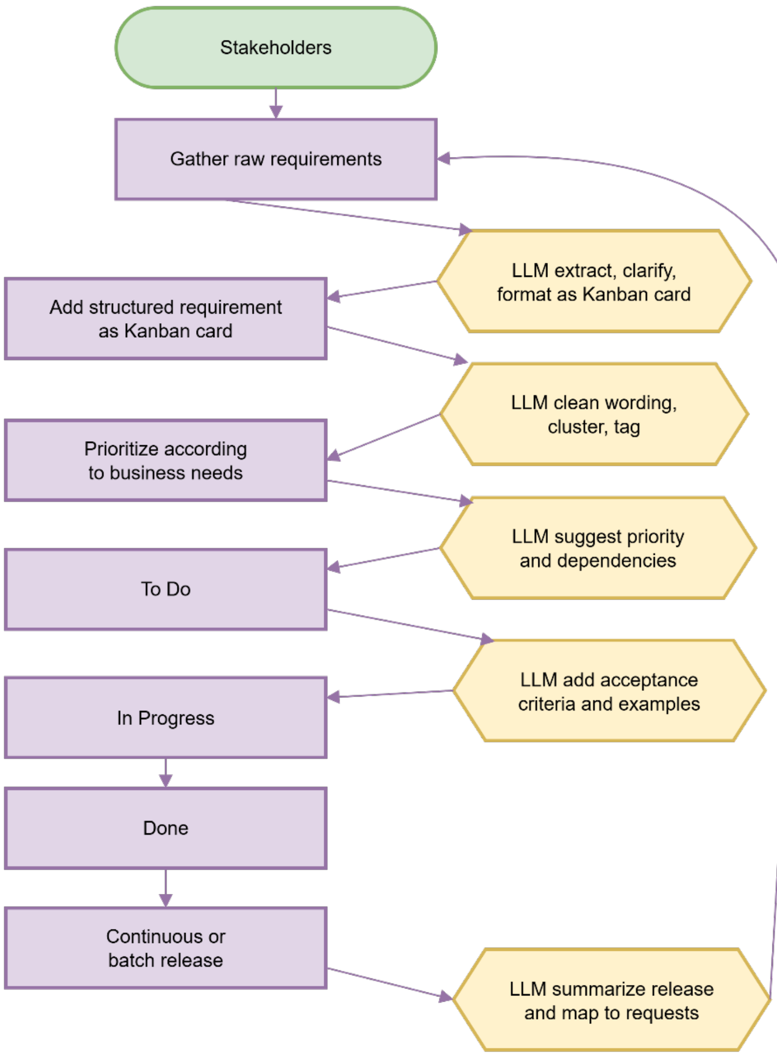

Stakeholders
Gather raw requirements
LLM extract, clarify, format as Kanban card
Add structured requirement as Kanban card
LLM clean wording, cluster, tag
Prioritize according to business needs
LLM suggest priority and dependencies
To Do
LLM add acceptance criteria and examples
In Progress
Done
Continuous or batch release
LLM summarize release and map to requests

Figure 7. LLM-Enhanced Kanban pipeline

LLM-Enhanced Kanban pipeline:

Stage 1.1 – Gather raw requirements

Stage 1.2 (LLM) - Before anything appears on the Kanban board, LLMs can process stakeholder inputs – from meeting notes, emails, or chat logs – to extract the core requirement, remove noise, and rewrite it as a concise, well-structured card candidate. They can also propose initial tags and detect missing details, generating clarifying questions for the product owner.

Stage 2.1 - Add structured requirement as Kanban card.

Stage 2.1 (LLM) - At this stage, the LLM can further clean wording, group similar cards, and ensure the descriptions follow the team's standards and the format is consistent.

Stage 3.1 - Prioritize according to business needs.

Stage 3.2 (LLM) - An LLM can review deadlines, business objectives, and dependency chains to propose a priority order. It might also highlight quick wins or warn about items with high complexity or risk.

Stage 4.1 - To Do.

Stage 4.2 (LLM) - Before work begins, the LLM can suggest acceptance criteria, generate example scenarios, and identify unclear requirements.

Stage 5 - In Progress.

Stage 6 - Done.

Stage 7.1 - Continuous or batch release.

Stage 7.2 (LLM) - When features are delivered, the LLM can summarize changes for release notes and trace delivered items back to their original requests or epics. This supports transparency, traceability, and compliance documentation.

Integrating LLMs into Kanban supports requirement gathering, card creation, prioritization, and continuous improvement. LLMs can extract and format stakeholder requests into structured Kanban cards, enrich them with tags and acceptance criteria, and assist in priority ordering based on business objectives and dependencies. During development, they help maintain clarity as work items progress and, after release, generate summaries and trace items back to original requests. This results in cleaner intake, faster prioritization, and improved traceability without disrupting Kanban's continuous flow.

4.3. LLM-enhanced Extreme Programming (XP) pipeline

While XP's lightweight artifacts encourage flexibility, they can be challenging to keep synchronized with evolving feedback and tests. By integrating Large Language Models, XP teams can accelerate story creation from customer conversations, expand acceptance tests with comprehensive scenarios, and maintain an up-to-date repository of requirements and tests. This helps preserve XP's agility while reducing the risk of overlooked or outdated requirements.

LLM-Enhanced XP pipeline:

Stage 1.1 – Planning Game: Customer presents requirements and priorities

Stage 1.2 (LLM) - LLM can normalize customer requirements, extract clear user stories following XP format, generate story estimates based on complexity patterns, and suggest release planning options. Create metaphors and system vocabulary to ensure shared understanding.

Stage 2.1 – Design Simply: Team creates simplest design

Stage 2.2 (LLM) – LLM can suggest simple design patterns, generate CRC cards automatically from user stories, identify potential over-engineering risks, and propose refactoring opportunities to maintain simplicity.

Stage 3.1 – Write Tests First: Create unit tests

Stage 3.2 (LLM) – LLM can auto-generate test cases from acceptance criteria, suggest edge cases and boundary conditions, create test data sets, and ensure comprehensive test coverage following TDD principles.

Stage 4.1 – Code in Pairs: Implement functionality

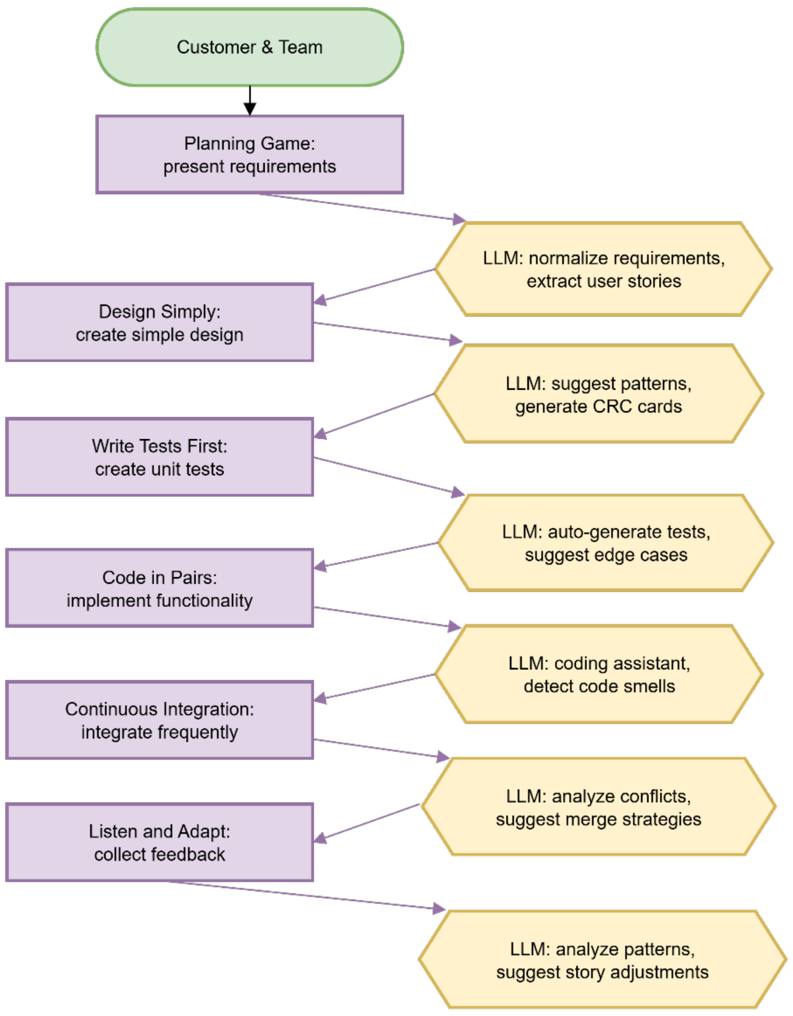
Stage 4.2 (LLM) – LLM can act as coding assistant during pair programming, suggest code improvements, detect code smells in real-time, and provide refactoring suggestions while maintaining collective code ownership.

Stage 5.1 – Continuous Integration: Integrate code frequently

Stage 5.2 (LLM) – LLM can analyze integration conflicts, suggest merge strategies, optimize build processes, and predict integration risks based on code changes.

Stage 6.1 – Listen and Adapt: Collect feedback from running software

Stage 6.2 (LLM) – LLM can analyze user feedback patterns, correlate feedback with specific user stories, suggest story adjustments, and identify emerging requirements for the next planning cycle.



Customer & Team
Planning Game: present requirements
LLM: normalize requirements, extract user stories
Design Simply: create simple design
LLM: suggest patterns, generate CRC cards
Write Tests First: create unit tests
LLM: auto-generate tests, suggest edge cases
Code in Pairs: implement functionality
LLM: coding assistant, detect code smells
Continuous Integration: integrate frequently
LLM: analyze conflicts, suggest merge strategies
Listen and Adapt: collect feedback
LLM: analyze patterns, suggest story adjustments

Figure 8. LLM-enhanced Extreme Programming (XP) pipeline

In Extreme Programming, LLMs can streamline the translation of customer conversations into structured user stories, propose thorough acceptance tests including edge cases, and keep these aligned through iterative feedback cycles. They help detect when tests or requirements become outdated, maintain traceability, and generate clear changelogs. The result is faster feedback incorporation, higher test coverage, and a continuously accurate repository of requirements and tests, while preserving XP's emphasis on close customer collaboration.

4.4. LLM-Enhanced SAFe Pipeline

While SAFe offers robust planning and traceability mechanisms, its complexity can introduce overhead in managing large volumes of requirements and dependencies. Integrating Large Language Models into SAFe can streamline decomposition, improve consistency in requirement descriptions, and enhance dependency management during Program Increment (PI) planning. LLMs can also strengthen traceability across all levels of the hierarchy, ensuring that business value is preserved from epic definition to feature delivery.

Stage 1.1 – Portfolio Strategy: Define strategic themes and business objectives

Stage 1.2 (LLM) - LLM can analyze market data, competitive intelligence, and business metrics to suggest strategic themes. Generate portfolio epic proposals based on business objectives, perform risk/benefit analysis, and map themes to value streams automatically.

Stage 2.1 – Solution Planning: Break down portfolio epics

Stage 2.2 (LLM) - LLM can decompose large portfolio epics into appropriately sized capabilities and program epics. Suggest capability boundaries, identify cross-ART dependencies, generate benefit hypotheses, and create traceability matrices between portfolio and program levels.

Stage 3.1 – Program Planning: PI Planning preparation and feature decomposition

Stage 3.2 (LLM) - LLM can assist in PI Planning by breaking capabilities into right-sized features, generating feature acceptance criteria, identifying program dependencies and risks, suggesting team capacity allocation, and creating draft PI objectives.

Stage 4.1 – Team Execution: Story creation and sprint execution

Stage 4.2 (LLM) - LLM can decompose features into implementable user stories following INVEST criteria, generate acceptance criteria and test cases, suggest story splitting techniques, provide estimation support, and identify blockers or dependencies between stories.

Stage 5.1 – Continuous Delivery: Release and feedback collection

Stage 5.2 (LLM) - LLM can analyze delivery metrics across ARTs, synthesize stakeholder feedback from inspect and adapt events, suggest process improvements, track value delivery against PI objectives, and recommend adjustments for future PI planning cycles.

Figure 9 displays LLM-Enhanced SAFe Pipeline.

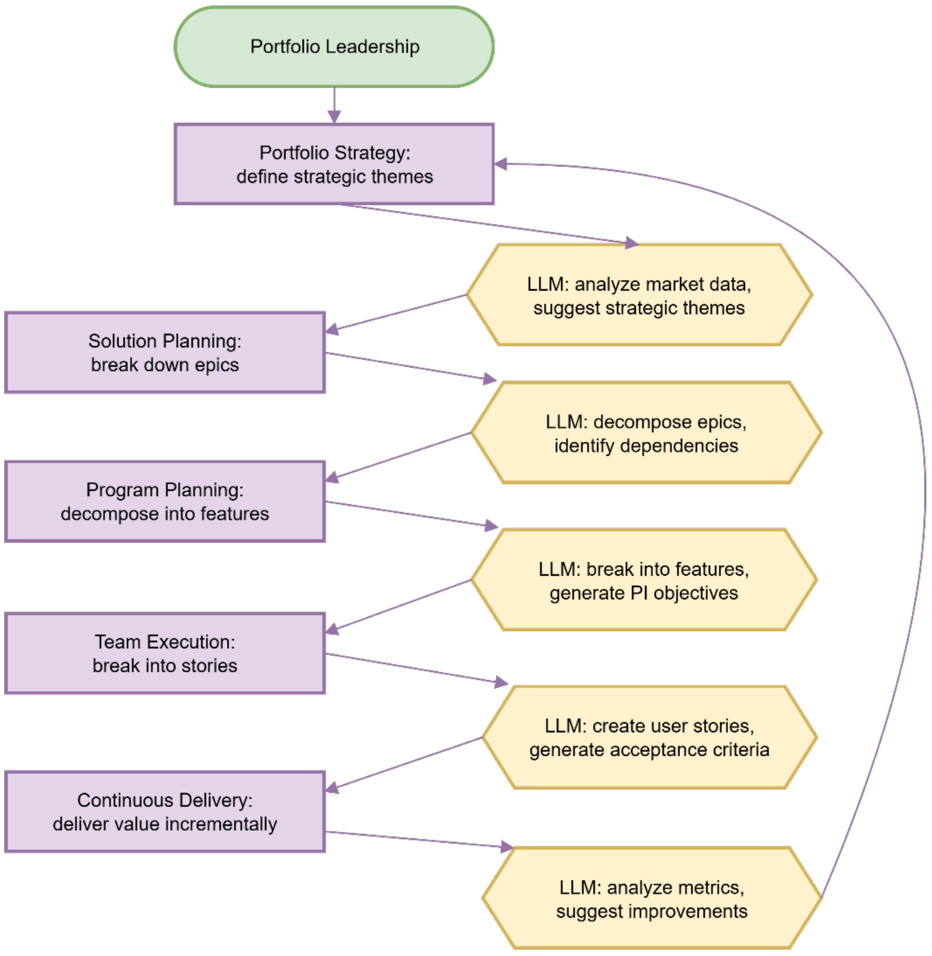

Portfolio Leadership
Portfolio Strategy: define strategic themes
LLM: analyze market data, suggest strategic themes
Solution Planning: break down epics
LLM: decompose epics, identify dependencies
Program Planning: decompose into features
LLM: break into features, generate PI objectives
Team Execution: break into stories
LLM: create user stories, generate acceptance criteria
Continuous Delivery: deliver value incrementally
LLM: analyze metrics, suggest improvements

Figure 9. LLM-Enhanced SAFe Pipeline

In SAFe, LLMs can assist at every level of requirement decomposition, from epics to user stories, ensuring alignment with business value and portfolio priorities. They help maintain consistency in acceptance criteria, visualize dependencies during PI planning, and preserve traceability across the entire hierarchy. The result is improved clarity, stronger cross-team coordination, and easier compliance reporting, all while supporting SAFe's structured, multi-level planning approach.

4.5. LLM-Enhanced Lean Software Development Pipeline

The minimalism that Lean Software Development provides makes this methodology highly adaptable, but it also risks omitting important details or traceability if not carefully managed. LLMs can enhance Lean by ensuring even lightweight requirements are clear, consistent, and traceable, without adding unnecessary bureaucracy.

Figure 10 displays modified Lean Software Development Pipeline with integrated LLMs stages.

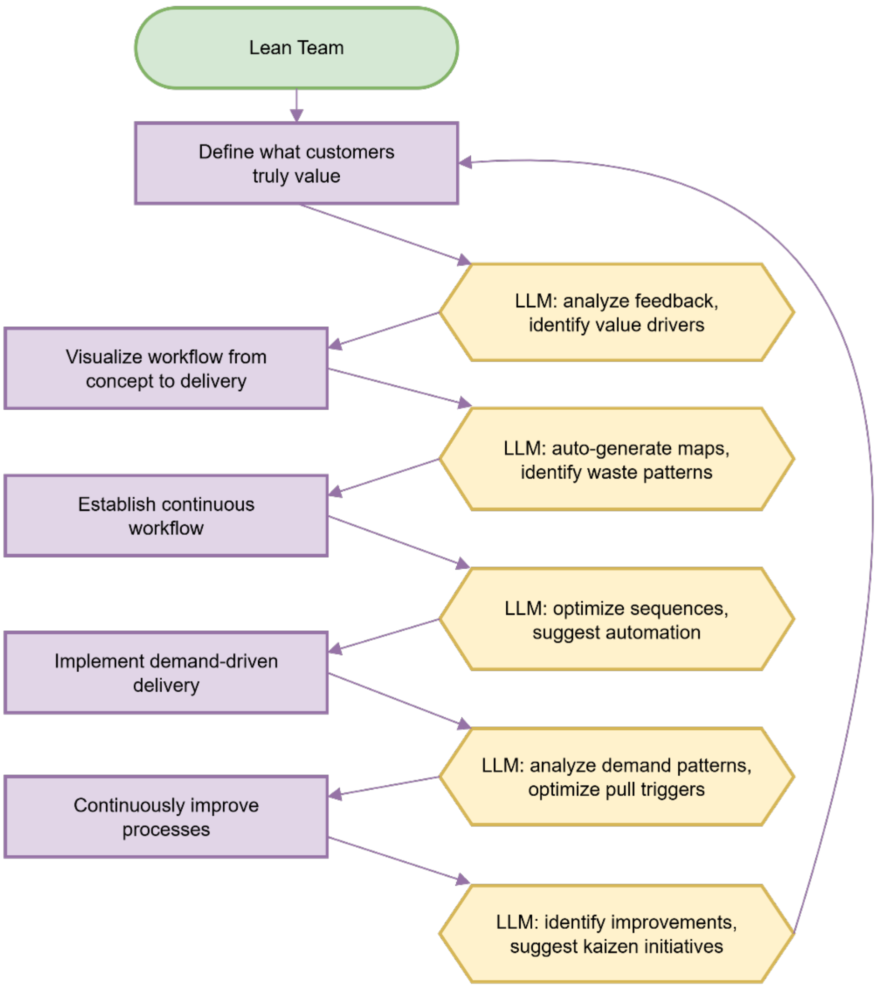

Lean Team
Define what customers truly value
LLM: analyze feedback, identify value drivers
Visualize workflow from concept to delivery
LLM: auto-generate maps, identify waste patterns
Establish continuous workflow
LLM: optimize sequences, suggest automation
Implement demand-driven delivery
LLM: analyze demand patterns, optimize pull triggers
Continuously improve processes
LLM: identify improvements, suggest kaizen initiatives

Figure 10. LLM-Enhanced Lean Software Development Pipeline

LLM-Enhanced Lean Software Development Pipeline:

Stage 1.1 - Define what customers truly value

Stage 1.2 (LLM) - LLM can analyze customer feedback, surveys, and usage data to identify true value drivers. Generate customer personas, prioritize value propositions, and suggest features that eliminate non-value-adding work automatically.

Stage 2.1 - Visualize workflow from concept to delivery

Stage 2.2 (LLM) - LLM can auto-generate value stream maps from process documentation, identify waste patterns across similar workflows, suggest bottleneck removal strategies, and predict where delays are likely to occur.

Stage 3.1 - Establish continuous workflow

Stage 3.2 (LLM) - LLM can optimize process sequences, suggest workflow automation opportunities, identify resource allocation improvements, monitor flow metrics in real-time, and recommend process standardization approaches.

Stage 4.1 - Implement demand-driven delivery

Stage 4.2 (LLM) - LLM can analyze demand patterns to predict customer needs, optimize pull system triggers, suggest capacity adjustments based on demand forecasting, and automate work initiation based on customer signals.

Stage 5.1 - Continuously improving processes

Stage 5.2 (LLM) - LLM can analyze performance metrics to identify improvement opportunities, suggest kaizen initiatives, synthesize feedback from multiple sources, track improvement impact, and recommend next optimization cycles.

This approach shows how AI enhances each Lean principle while maintaining the focus on waste elimination and continuous flow that defines Lean methodology.

4.6. Chapter summary

LLMs can summarize user reactions, identify common improvement themes, and propose next-step requirements. This helps ensure the next iteration is informed by clear, structured insights instead of raw, unfiltered feedback.

This chapter examined how Large Language Models can be embedded into the requirements management processes of five Agile methodologies: Scrum, Kanban, Extreme Programming (XP), the Scaled Agile Framework (SAFe), and Lean Software Development. For each methodology, we mapped LLM capabilities directly to its native workflow stages, introduced specific enhancement opportunities, and illustrated them with annotated process diagrams.

Across all frameworks, LLMs were shown to add value at key points such as requirement intake, backlog or board preparation, prioritization, acceptance criteria generation, and post-delivery feedback analysis. The depth of integration varied according to the methodology: Scrum and SAFe benefit from structured decomposition and traceability support; Kanban gains from improved card clarity and prioritization; XP from enriched acceptance testing and synchronization with evolving feedback; and Lean from structured yet lightweight documentation that preserves agility while reducing waste.

A consistent pattern emerged – LLMs can reduce ambiguity, accelerate refinement, and strengthen traceability without disrupting the core principles of each methodology. However, the chapter also reinforced that these benefits depend on maintaining human oversight, tailoring integration to the workflow, and ensuring alignment with organizational priorities and compliance requirements.

5. Conclusion

The integration of Large Language Models into Agile requirements management offers a promising avenue for enhancing efficiency, consistency, and traceability across diverse development contexts. By aligning LLM capabilities with the distinct workflows of Scrum, Kanban, Extreme Programming, SAFe, and Lean Software Development, this study has shown that AI can effectively support both structured and lightweight Agile approaches.

LLMs excel in transforming unstructured stakeholder input into well-formed requirements, enriching backlog items or work cards with acceptance criteria, identifying dependencies, and maintaining traceability across the development lifecycle. They also facilitate faster refinement cycles, more informed prioritization, and clearer post-delivery feedback analysis. While the level and style of integration vary by methodology, the underlying benefits are consistent: reduced ambiguity, improved requirement quality, and more efficient collaboration between business and technical stakeholders.

However, the deployment of LLMs in requirements engineering is not without risks. Challenges such as hallucinations, bias, over-reliance on automation, and data privacy concerns underscore the need for robust governance and human-in-the-loop oversight. Moreover, integration strategies must be tailored to each methodology's principles to avoid undermining the very agility they aim to support.

Future research should explore domain-specific LLM fine-tuning, evaluate real-world productivity impacts over extended projects, and investigate hybrid human–AI collaboration models that optimize both speed and accuracy. As technology matures, LLM-assisted requirements management has the potential to become an integral part of Agile practice, bridging the gap between human creativity and machine efficiency.

Declaration on Generative AI

The author(s) have not employed any Generative AI tools.

References

[1] Kharchenko O., Yatsyshyn V. Rozrobka ta keruvannya vymohamy do prohramnoho zabezpechennya na osnovi modeli yakosti. Visnyk TDTU. 2009. Tom 14. №1. S. 201-207.

[2] Pastukh O., Yatsyshyn V. Development of software for neuromarketing based on artificial intelligence and data science using high-performance computing and parallel programming technologies. Scientific Journal of TNTU. Tern.: TNTU, 2024. Vol 113. No 1. P. 143–149.

[3] Yatsyshyn V, Kharchenko A, Galay I. The Method of Quality Management Software. Proceeding of the VIIth International Conference "Perspective technologies and methods in MEMS design" 11-14 May 2011 – Polyana, Ukraine: Publishing House Vezha&Co. 2011.- p. 228-230.

[4] ISO/IEC 14598: Information Technology – Software product evaluation. Parts 1 to 6: 1999 -2001, International Organization for Standardization, Geneva, 1999-2001.

[5] Schwaber, K., & Sutherland, J. (2020). The Scrum Guide. The Definitive Guide to Scrum: The Rules of the Game. Scrum.org.

[6] Rubin, K. S. (2012). Essential Scrum: A Practical Guide to the Most Popular Agile Process. Addison-Wesley Professional.

[7] Anderson, D. J. (2010). Kanban: Successful Evolutionary Change for Your Technology Business. Blue Hole Press.

[8] Ahmad, M. O., Markkula, J., & Oivo, M. (2013). Kanban in software development: A systematic literature review. 39th Euromicro Conference on Software Engineering and Advanced Applications, 9-16.

[9] Wells, D. (2013). Extreme Programming: A gentle introduction. ExtremeProgramming.org.

[10] Leffingwell, D. (2018). SAFe 4.5 Reference Guide: Scaled Agile Framework for Lean Enterprises. Addison-Wesley Professional.

[11] Knaster, R., & Leffingwell, D. (2017). SAFe 4.0 Distilled: Applying the Scaled Agile Framework for Lean Software and Systems Engineering. Addison-Wesley Professional.

[12] Poppendieck, M., & Poppendieck, T. (2003). Lean Software Development: An Agile Toolkit. Addison-Wesley Professional.

[13] Brown, T., Mann, B., Ryder, N., et al. (2020). Language models are few-shot learners. Advances in Neural Information Processing Systems, 33, 1877-1901.

[14] Vaswani, A., Shazeer, N., Parmar, N., et al. (2017). Attention is all you need. Advances in Neural Information Processing Systems, 30, 5998-6008.

[15] Ji, Z., Lee, N., Frieske, R., et al. (2023). Survey of hallucination in natural language generation. ACM Computing Surveys, 55(12), 1-38.

[16] Chen, M., Tworek, J., Jun, H., et al. (2021). Evaluating large language models trained on code. arXiv preprint arXiv:2107.03374.

[17] Wang, D., Churchill, E., Maes, P., et al. (2020). From human-human collaboration to human-AI collaboration: Designing AI systems that can work with people. Proceedings of the 2020 CHI Conference Extended Abstracts, 1-6.

[18] Semchyshyn, V., & Mykhalyk, D. (2023). Automated processing and analysis of medical texts. ITTAP, 300-305.

[19] Lypak, O.H., Lytvyn, V., Lozynska, O., Rzheuskyi, A., Dosyn, D. (2019). Formation of Efficient Pipeline Operation Procedures Based on Ontological Approach. Advances in Intelligent Systems and Computing, 871, 571–581.