Article

Using AI Agents in Software Requirements

James Helfrich * and Emilio Regino

Posted Date: 31 October 2025

doi: 10.20944/preprints202510.2470.v1

Keywords: AI; agent; requirements engineering; LLM

 doi:10.20944/preprints202510.2470.v1

Article

Using AI Agents in Software Requirements

James Helfrich * and Emilio Regino

Brigham Young University

* Correspondence: HelfrichJ @byui.edu; Tel.: +1 208 496 7608

Abstract

Requirements engineering (RE) plays a pivotal role in the software development lifecycle, providing the foundation for all downstream design, implementation, and validation activities. However, RE is inherently challenging due to the sheer volume and complexity of elicitation data. Fatigue, selective attention, and unconscious bias undermine results by human agents. We present a training method for Artificial Intelligence (AI) agents that systematically transforms raw elicitation data into high quality requirements without susceptibility to these limitations. With a corpus of 28 stakeholder interviews, requirements were produced by human analysts, baseline (untrained) AI agents, prompt engineered AI agents, and agents trained with our method. An independent panel of experts rated completeness, consistency, traceability, and stakeholder alignment. Trained agents significantly outperformed both humans and untrained agents across all criteria, delivering broader coverage and fewer conflicts while maintaining format fidelity. These results indicate that purpose trained AI agents can raise RE quality and throughput beyond current practice, offering a scalable, auditable pipeline from elicitation to specification.

Keywords: AI; agent; requirements engineering; LLM

1. Introduction

Requirements Engineering is defined as "the process of defining, documenting, and maintaining requirements" [1]. Brooks stated, "the hardest single part of building a software system is deciding precisely what to build. No other part of the conceptual work is as difficult as establishing the detailed technical requirements, including all the interfaces to people, to machines, and to other software systems. No other part of the work so cripples the resulting system if done wrong" [2], p.20. Since the public release of ChatGPT, there has been widespread adoption of AI tools in the software development process. A recent survey of 33,662 participants showed that 84% of all developers use AI tools in their practice and a further 5% plan to adopt them [3]. Despite these advances, there has been little research exploring the applicability of AI tools to requirements engineering. This paper explores this gap in the literature with three research questions:

RQ1: What is the relative effectiveness of different AI prompting strategies for generating requirements, use cases, and user stories?

RQ2: How does AI-developed requirements, use cases, and user stories compare with those produced by human agents?

RQ3: Does the choice of large language model (LLM) technology used by an AI agent affect the quality of the requirements, use cases, and user stories?

2. Literature Review

There are three relevant lines of research: agent engineering, requirements engineering and the use of large language models (LLMs) to complete requirements engineering tasks.

 doi:10.20944/preprints202510.2470.v1

2.1. Agent Engineering

The term *prompt engineering* was coined by Gwern Branwen in his 2020 essay *Prompts and Programming* [4], though it did not appear in an academic paper until 2021 [5]. Prompt engineering is "strategically designing task-specific instructions, referred to as prompts, to guide model output without altering parameters" [6]. Several prompt engineering techniques have been described, including Chain-of-Thought [7], Logical Chain-of-Thought [8], Chain of Symbol [9], Automatic Chain-of-Thought [10], Least-to-Most [11], Self-Consistency prompting [12], and Tree-of-Thoughts [13]. The common characteristic is to provide all the examples, context, and specificity required for an LLM to produce the desired outcome [14].

A more advanced form of AI is the use of AI agents [15] or agentic systems [16], defined as systems that perceive their environment and act to maximize expected performance based on prior knowledge and experience [17]. Building such systems, known as Agent Engineering [18], involves "the development of autonomous computational or physical entities capable of perceiving, reasoning, adapting, learning, cooperating and delegating in a dynamic environment." Several agents have been developed for specific tasks, such as software engineering [19], e-commerce navigation [20], and even for Minecraft tasks [21] Techniques used to train AI agents for these specialized tasks include imitation learning [20], reinforcement learning [21], task-specific fine-tuning [22], and system-level scaffolding [19]. As Shah summarizes: "We also need this [agent] to be systematically created with enough generalizability and explainability to be useful for the same or similar experiments by other researchers, by other LLMs, and at different times" [23], p.57.

2.2. Requirement Engineering

The term *Requirement Engineering* was first introduced in 1969 [24], and formalized in the software engineering lifecycle models by Royce the next year [25]. In 1984, IEEE published IEEE 830-1984, which formalized the Software Requirements Specification (SRS) and outlined recommended practices for its structure and content [26]. The central feature of a SRS is a software requirement, defined as "a condition or capability needed by a user to solve a problem or achieve an objective" [27]. IEEE defines ten attributes of well-formed requirements: necessary, implementation-free, unambiguous, consistent, complete, singular, feasible, traceable, verifiable, bounded [28]. An empirical study found that ambiguity, completeness, consistency, and correctness have the largest impact on the quality of software requirements [29]. Since then, Bell and Thayer criticized the practice of utilizing heavy, document-centric requirements process and advocated for new methodologies more attuned to stakeholder needs [30].

In 1987, Ivar Jacobson was the first to introduce an alternative to requirements—use cases—during a presentation at the OOPSLA '87 conference [31]. Jacobson later defined a use case as "all the ways of using a system to achieve a particular goal for a particular user" [32]. Use cases experienced widespread attention with Jacobson's 1992 *book Object-Oriented Software Engineering: A Use Case Driven Approach* [33]. In 1995, Carroll introduced scenarios, framed as specific instances or examples how a use case might play out in practice [34]. The final and most influential variation of requirements was introduced by Kent Beck in 1997: user stories [35]. A user story is "one thing the customer wants the system to do. … Stories should be testable" [35, p. 179]. Other formats designed to capture system needs include GRAIL/KAOS [36], i* [37], FODA [38], and Z notation [39]. In practice today, user stories are the most commonly used format to represent software needs—employed by about 45% of agile practitioners, while traditional requirements remain the second most common [40].

2.3. LLMs and RE

The first study exploring applicability of utilizing LLMs in RE processes was conducted in 2023. Ronanki, Berger, and Horkoff found that ChatGPT-generated requirements were highly abstract, atomic, consistent, correct, and understandable, though lacking in aspects such as unambiguity and feasibility [41]. Rahman and Zhu introduced a tool *GeneUS*, which uses GPT-4 to generate user stories

 doi:10.20944/preprints202510.2470.v1

from elicitation data [42]. Sami et al. implemented a multi-agent approach, deploying four LLMs to generate user stories from four real-world projects; they found that each model performed with varying degrees of detail and response time [43]. Finally, Hymel and Johnson reported that LLM-generated requirements were more aligned and complete than human-generated ones [44]. Currently, no studies compare the quality of requirements, user stories, or use cases across multiple AI prompting techniques.

3. Materials and Methods

The experimental setup consisted of the collection of elicitation data, generation of system needs from naïve prompts, generation of system needs from engineered prompts, generation of system needs from AI agents, and the subsequent analysis.

3.1. Elicitation Data

To test the capacity of AI tools to perform requirements engineering tasks, elicitation data was collected for a new software system: a mobile application designed to help users track their habits. This elicitation data consisted of 246 minutes of interviews with 28 potential users, as well as nine competitive analysis reports. The elicitation data was carefully scrubbed of personally identifiable information (PII) by replacing names with randomly generated numbers.

3.2. Naïve Prompts

The first set of prompts was created lacking any specific prompt engineering techniques. The prompt for requirements was the following: "Create a software requirements specification from the provided elicitation data." The prompt for user stories was the same as that for requirements except "a complete set of user stories" replaced "a software requirements specification." A similar substitution was applied to use cases.

3.3. Prompt Engineering

The second set of prompts was created by leveraging best practices identified in the literature. These prompts emphasize specificity (task, interfaces, and constraints) and context. The prompt to generate use cases was as follows: "You are a requirements engineer. You are known for being thorough and detailed in your analysis of elicitation data. Your task is to create a Product Backlog consisting of a complete set of user stories from the provided data. Each user story must describe the system functionality from the user's perspective in the form: 'As a [user], I want [functionality] so that [benefit].' As you review the elicitation data, identify the users, the anticipated benefits, and the functionality required to deliver those benefits. Begin with epic user stories and then subdivide them into large, medium, and eventually small stories. When finished, the set of small user stories must comprehensively cover all user needs as specified in the elicitation data."

The prompt to generate user stories was as follows: "You are a requirements engineer. You are known for being thorough and detailed in your analysis of elicitation data. Your task is to create a Product Backlog consisting of a complete set of User Stories from provided data. Each user story will describe the system functionality from the user's perspective. Each user story will be in the form of 'as a [user], I want [functionality] so that [benefit].' As you look through the elicitation data, identify the users, the benefit the user anticipates, and the functionality that will deliver the user this benefit. Start with epic user stories and then sub-divide them into large, then medium, and eventually small. When finished, the SRS must comprehensively cover all user needs as specified in the elicitation data."

The prompt to generate an SRS was as follows: "You are a requirements engineer. You are known for being thorough and detailed in your analysis of elicitation data. Your task is to generate a software requirements specification (SRS) from the provided data. The SRS must consist of a set of definitions and requirements. Each requirement should describe necessary system functionality from

 doi:10.20944/preprints202510.2470.v1

the system's perspective. When two or more requirements reference the same entity, define that entity in the definitions section. Each requirement must be clear, concise, complete, correct, consistent, and unambiguous to ensure that the software meets the needs of all stakeholders. When finished, the SRS must comprehensively cover all user needs as specified in the elicitation data."

3.4. Agent Engineering

The third set of prompts involved agent engineering, where the LLM was carefully trained to perform specific tasks. In this case, the tasks were to generate an SRS, a product backlog, and a complete set of use cases. The training was implemented using markdown instruction files. Each markdown file contained four components:

Role: A multi-sentence description of the role the LLM was to play in the requirements elicitation process.

Task: The task consists of a detailed description of every component of the process that the LLM was to perform. This largely consisted of a codebook, a "finite (and preferably small) set of labels and their desired descriptions" [23], p. 56. In the case of the SRS instruction file, the labels include SRS, system, elicitation data, interviews, observations, competitive analysis data, stakeholder, user, customer, support, definitions, requirement list, quality, complete, consistent, modifiable, traceable, requirement, functional requirement, non-functional requirement, correct, feasible, necessary, prioritized, unambiguous, and verifiable.

Steps: The process the LLM was to perform to complete the task. In the case of the use case instruction file, the steps consisted of identifying actors, describing the system, and describing the action.

Examples: The examples consists of sample elicitation data and the resulting requirements, use cases, or user stories.

3.5. Data Collection and Analysis

Requirements, user stories, and use cases generated by human agents, naïve prompts, engineered prompts, and AI agents were assigned the labels listed in Table 1.

Table 1. List of labels corresponding to requirements, use cases, and user stories created by human experts, with naïve prompts, with engineered prompts, and with AI agents.

	Human	Naïve Prompt	Prompt Engineer	AI Agent
SRS	SRS-Human	SRS-Naïve	SRS-Prompt	SRS-Agent
Use Case	UseCase-Human	UseCase-Naïve	UseCase-Prompt	UseCase-Agent
User Story	UserStory-Human	UserStory- Naïve	UserStory-Prompt	UserStory-Agent

Each research question necessitated the ability to determine the relative quality of two requirement sets. This was performed by computing a coverage score and a quality score.

Coverage was determined through two trained experts analyzing each set, correlating a requirement in one set with those of the others. The total number of unique requirements in the available sets (SRS-Human, SRS-Naïve, SRS-Prompt, and SRS-Agent) is called Num-SRS-Total, with Num-SRS-Human being the number of human-found requirements and so on. The coverage score for human requirements is Coverage-SRS-Human = Num-SRS-Human / Num-SRS-Total where $0 \leq$ Coverage-SRS-Human ≤ 1.

Quality was determined by verifying correctness, feasibility, necessity, ambiguity, and verifiability for each individual requirement according to the IEEE specification with two trained experts. The number of defective requirements is Defects-SRS-Human. The quality score for human requirements is Quality-SRS-Human = (Num-SRS-Human – Defects-SRS-Human) / Num-SRS-Human where $0 \leq$ Quality-SRS-Human ≤ 1.

To address RQ1, SRS-Naïve was compared against SRS-Prompt and SRS-Agent using the coverage and quality criteria. This was accomplished by comparing Coverage-SRS-Naïve with

 doi:10.20944/preprints202510.2470.v1

Coverage-SRS-Prompt and Coverage-SRS-Agent, as well as Quality-SRS-Naïve with Quality-SRS-Prompt and Quality-SRS-Agent. The same was done with use cases and user stories.

To address RQ2, SRS-Agent was compared against SRS-Human using the same coverage and quality criteria. The same was done with use cases and user stories.

To address RQ3 for requirements, the same training file was utilized with ChatGPT 5.0, ChatGPT 4o, Claude, Gemini, and Grok. The resulting requirements were analyzed using the same coverage and quality criteria applied to RQ1 and RQ2. The same was applied to user stories and use cases.

4. Results

4.1. RQ1

The first research question is: "What is the relative effectiveness of different AI prompting strategies for generating requirements, use cases, and user stories?" To evaluate this question, nine data sets were generated: SRS-Naïve, SRS-Prompt, SRS-Agent, UserStory-Naïve, UserStory-Prompt, UserStory-Agent, UseCase-Naïve, UseCase -Prompt, and UseCase-Agent. The results are presented in Table 2.

Table 2. Coverage and Quality score for requirements, user stories, and use cases generated using naïve, prompt engineering, and agent engineering strategies.

Set	Coverage	Quality
SRS-Naïve	0.86	0.54
SRS-Prompt	0.74	0.46
SRS-Agent	0.91	0.75
UserStory-Naïve	0.40	0.31
UserStory-Prompt	0.83	0.25
UserStory-Agent	0.83	0.57
UseCase-Naïve	0.86	0.50
UseCase-Prompt	0.91	0.53
UseCase-Agent	0.91	0.54

Two experts analyzed each requirement and found all were supported by elicitation data, though the lack of citations made verification difficult in SRS-Naïve and SRS-Prompt. There were four requirements that SRS-Prompt and SRS-Agent identified that SRS-Naïve missed, yielding a decreased score for coverage. The same expert analyzed each requirement from a quality standpoint. SRS-Naïve lacked a terms section so terms like "milestone" and "icon" were not fully defined. SRS-Prompt and SRS-Agent included a complete definitions/terms section, and no ambiguity was found in either set. In summary, the expert found SRS-Agent to be the most comprehensive SRS, including detailed definitions, priority levels for each requirement, explicit source citations, and stricter adherence to formal requirement phrasing. Thus, the agent engineering strategy produced significantly higher-quality requirements than prompt engineering or naïve prompting strategies. The same analysis was performed on user stories. 141 unique user stories were identified, with 22 found only in UserStory-Naïve, 36 found in both UserStory-Prompt and UserStory-Agent but not in UserStory-Naïve, and the remainer found in all three. From a quality perspective, UserStory-Agent and UserStory-Prompt were meticulously organized into epics, large, medium, and small user stories, providing a clear hierarchy and breakdown of functionality. UserStory-Naïve, while comprehensive, presented a flatter list of stories, categorized only by broad functional areas without further sub-division or explicit sizing. UserStory-Agent and UserStory-Prompt consistently followed the "As a [Actor], I want [Action] so that [Motivation/Benefit]" template, producing highly consistent and clear statements of user needs. UserStory-Naïve largely follows this template but uses slightly less precise phrasing or embedded contextual details within the story itself. Thus, the prompt engineering and agent engineering strategies significantly outperformed the naïve strategy in terms

 doi:10.20944/preprints202510.2470.v1

of completeness, while the agent engineering strategy slightly outperformed the prompt engineering strategy for quality. The same analysis was performed on use cases. A total of 76 unique use cases was identified, with 2 found only in UseCase-Naïve, 2 found only in UseCase-Prompt, 3 found only in UseCase-Agent, 4 found not in UseCase-Agent but in the other two, and one not found in UseCase-Naïve but found in the other two. A more detailed look found that all the core functionality was present in all three sets. UseCase-Agent employed more formal and less ambiguous language throughout. The difference in prompting techniques was not as pronounced with use cases as they were with user stories and, especially, requirements.

4.2. RQ2

The second research question is: "How does AI-developed requirements, use cases, and user stories compare with those produced by human agents?" To evaluate this question, 6 data sets were analyzed: SRS-Human, SRS-Agent, UserStory-Human, UserStory-Agent, UseCase-Human, and UseCase-Agent. The results are presented in Table 3.

Table 3. Coverage and Quality score for requirements, user stories, and use cases generated using naïve, prompt engineering, and agent engineering strategies.

Set	Coverage	Quality
SRS-Human	0.17	0.60
SRS-Agent	0.91	0.75
UserStory-Human	0.44	0.18
UserStory-Agent	0.83	0.57
UseCase-Human	0.82	0.50
UseCase-Agent	0.91	0.54

Two experts analyzed each requirement, finding SRS-Agent provides "a far broader … set of requirements, encompassing a wide array of user needs and advanced features." They described as SRS-Human as "a very basic set of requirements." Similarly, one expert reported UserStory-Agent captured "detailed functionalities and nuanced user needs identified in the interviews and competitive analyses. It elaborates on complex features, advanced customization, specific motivational elements, detailed reporting options, and various external integrations." UserStory-Human was described as "high-level" and "flatter" and "primarily focuses on basic functionalities." UseCase-Agent was described as containing "elaborate basic flows" and "numerous alternate flows." The same experts described UseCase-Human as "minimalistic" and "simple."

4.3. RQ3

The third research question is: "Does the choice of large language model (LLM) technology used by an AI agent affect the quality of the requirements, use cases, and user stories?" To evaluate this question, nine data sets were generated using agent engineering training techniques: SRS-GPT50, SRS-GPT35, SRS-Claude, SRS-Gemini, UserStory-GPT50, UserStory-GPT35, UserStory-Claude, UserStory-Gemini, UseCase-GPT50, UseCase-GPT35, UseCase-Claude, and UseCase-Gemini. The results are presented in Table 4.

Table 4. Coverage and quality score for requirements, user stories, and use cases generated using various ChatGPT 4o, ChatGPT 5.0, Claude, and Gemini.

Set	Coverage	Quality
SRS-GPT4o	0.59	0.73
SRS-GPT5.0	0.55	0.73
SRS-Claude	0.84	0.77
SRS-Gemini	0.66	0.84

 doi:10.20944/preprints202510.2470.v1

UserStory-GPT4o	0.65	0.27
UserStory-GPT5.0	0.72	0.27
UserStory-Claude	0.74	0.57
UserStory-Gemini	0.85	0.17
UseCase-GPT4o	0.53	0.40
UseCase-GPT5.0	0.77	0.96
UseCase-Claude	0.77	0.83
UseCase-Gemini	0.16	0.00

Gemini received the lowest quality scores for user stories and use cases but performed strongly on requirements. ChatGPT-4o achieved results comparable to ChatGPT-5.0 for requirements and user stories; however, it was substantially outperformed on use cases.

5. Discussion

The first research question is: "What is the relative effectiveness of different AI prompting strategies for generating requirements, use cases, and user stories?" Across all three artifacts, there is a clear trend indicating that more sophisticated prompting strategies yield better results. Prompt engineering scored higher on coverage and higher quality for user stories and use cases but performed slightly worse for requirements. The most significant improvements occurred in the transition from prompt engineering to agent engineering.

The second research question is: "How do AI-developed requirements, use cases, and user stories compare with those produced by human agents?" In terms of coverage, agent-assisted AI substantially outperformed human authors. When asked, the human authors cited boredom, time constraints, and the tendency to overlook details in the elicitation data as the reason for overlooking subtle requirements. In terms of quality, agent-assisted AI outperformed human authors for requirements and user stories but performed comparably on use cases.

The third research question is: "Does the choice of large language model (LLM) technology used by an AI agent affect the quality of the requirements, use cases, and user stories?" Although differences were observed across the four models, the choice of LLM had less impact on coverage and quality than the prompting technique.

This study faced two major limitations. The first was the variability in outputs from the different LLMs: executing the same prompt twice often produced noticeably different results. This limitation could have been mitigated by conducting multiple executions and analyzing each result systematically. The second limitation was the high cost of having human evaluators analyze each requirement, product backlog, and set of use cases. In total, 24 sets were generated and analyzed in this study. A more objective and scalable approach would be to develop a custom AI agent to perform this mechanical evaluation task.

One potential area for future research is the exploration of different agent engineering techniques. At present, little work has been done to categorize or describe the tools and methods in this space. An empirical study that isolates each of these techniques to determine their impact on the suitability of AI agents for performing requirements engineering tasks would be highly beneficial.

Abbreviations

The following abbreviations are used in this manuscript:

AI	Artificial Intelligence
LLM	Large language model
PII	Personally identifiable information
RE	Requirements engineering
SRS	Softwar e requirements specification

 doi:10.20944/preprints202510.2470.v1

References

1. Kotonya, G. and Sommerville, I. *Requirements Engineering: Processes and Techniques*. s.l. : John Wiley & Sons, Inc., 1998.
2. Brooks, Frederick. *Mythical Man-Month, The: Essays on Software Engineering*, Anniversary Edition. s.l. : Addison-Wesley Professional, 1995.
3. Stack Overflow. *2025 Developer Survey*. Stack Overflow. [Online] 2025. https://survey.stackoverflow.co/2025/ai.
4. Branwen, Gwern. *GPT-3 Creative Fiction*. Gwern.net. [Online] 06 19, 2020. https://gwern.net/gpt-3#sn5.
5. Liu, Vivian and Chilton, Lydia. Design Guidelines for Prompt Engineering Text-to-Image Generative Models. CHI Conference on Human Factors in Computing Systems, New Orleans : ACM, 2021. pp. 1-23.
6. Sahoo, P., Singh, A. K., Saha, S., & Jain, V. A Systematic Survey of Prompt Engineering in Large Language Models: Techniques and Applications. 2024, arXiv preprint arXiv:2402.07927.
7. Wang, Xuezhi, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. *Self-consistency improves chain of thought reasoning in language models*. 2022, arXiv, p. arXiv preprint arXiv:2203.11171.
8. Xhao, X., Li, M., Lu, W., Weber, C., Lee, J. H., Chu, H., & Wermter, S. *Enhancing Zero-Shot Chain-of-Thought Reasoning in Large Language Models through Logic*. arXiv. 2023.
9. Hu, H., Lu, H., Zhang, H., Song, Y.-Z., Lam, W., & Zhang, Y. *Chain-of-symbol prompting elicits planning in large language models*. arXiv. 2024.
10. Zhang, Z., Zhang, A., Li, M., & Smola, A. Automatic chain of thought prompting in large language models. 2022, arXiv.
11. Zhou, D., Schärli, N., Hou, L., Wei, J., Scales, N., & Wang, X. *Least-to-most prompting enables complex reasoning in large language models*. 2022, arXiv.
12. Wang, Xuezhi, et al. Self-Consistency Improves Chain of Thought Reasoning in Language Models. ICLR 2023.
13. Yao, S., Yu, D., Zhao, J., Shafran, I., Griffiths, T., & Cao, Y. N. *Tree of thoughts: Deliberate problem solving with large language models*. 2023.
14. Brown, T. et al. Language Models are Few-Shot Learners. Advances in neural information processing systems 33. 2020, pp. 1877-1901.
15. Castelfranchi, Cristiano. *Modelling Social Action for AI Agents* 1-2, 1998, Artificial Intelligence, Vol. 103, pp. 157-182.
16. Anthropic. *Building Effective Agents. Anthropic*. [Online] 12 19, 2024. https://www.anthropic.com/engineering/building-effective-agents.
17. Russell, Stewart and Norvig, Peter. *Artificial Intelligence: A Modern Approach*. s.l. : Prentice Hall, 2022.
18. Liu, Jiming. *Agent Engineering*. s.l. : World Scientific, 2001.
19. Yang, J., Jimenez, C., Wettig, A., Lieret, K., Yao, S., Narasimhan, K., & Press, O. SWE-Agent: Agent-Computer Interfaces Enable Automated Software Engineering. 2024. *Advances in Neural Information Processing Systems* 37 (NeurIPS 2024). pp. 50528-50652.
20. Yao, S., Chen, H., Yang, J., & Narasimhan, K. WebShop: *Towards Scalable Real-World Web Interaction with Grounded Language Agents*. 2022. 36th Conference on Neural Information Processing Systems. pp. 20744-20757.
21. Fan, L., et al. Minedojo: Building open-ended embodied agents with internet-scale knowledge. 2022. *Advances in Neural Information Processing Systems*. pp. 18343-18362.
22. Chebotar, Yevgen et al. RT-2: Vision-Language-Action Models Transfer Web Knowledge to Robotic Control. 2023. Conference on Robot Learning. pp. 2165-2183.
23. Shah, Chirag. From Prompt Engineering to Prompt Science with Humans in the Loop. 6, 2025, *Communications of the ACM*, Vol. 68, pp. 54-61.
24. Naur, Peter and Randell, Brian. Software Engineering: *Report of a conference sponsored by the NATO Science Committee*. Brussels : Scientific Affairs Division, NATO, 1969.
25. Royce, Winston. Managing the development of large software systems. 1970, Proceedings of the IEEE WESCON, Vol. 26, pp. 1-9.

 doi:10.20944/preprints202510.2470.v1

26. IEEE. IEEE Guide for Software Requirements Specifications. s.l. : IEEE, 1984.
27. Washizaki, H. Guide to the Software Engineering Book of Knowledge (SWEBOK). s.l. : IEEE Computer Society, 2025.
28. International Organization of Standardization. ISO/IEC/IEEE 29148:2011 - Software Systems and Software Engineering - Life Cycle Processes - Requirements Engineering. 2011.
29. Montgomery, L., Fucci, D., Bouraffa, A., Scholz, L., & Maalej, W. Empirical research on requirements quality: a systematic mapping study. 2, 2022, *Requirements Engineering*, Vol. 27, p. 1830209.
30. Bell, Thomas and Thayer, Thomas. Software requirements: Are they really a problem? 1976. Proceedings of the 2nd international conference on Software engineering. pp. 61-68.
31. Jacobson, Ivar. Object Oriented Development in an Industrial Environment. 1987. OOPSLA '87 Proceedings. pp. 183-191.
32. Jacobson, Ivar, Spence, Ian and Kerr, Brian. *Use Case Definition*. Ivar Jacobson International. [Online] January 2016. https://www.ivarjacobson.com/publications/white-papers-articles/use-case-definition.
33. Jacobson, Ivar. Object-Oriented Software Engineering: A Use Case Driven Approach. s.l. : Addison-Wesley, 1992.
34. Carroll, John. Scenario-Based Design: Envisioning Work and Technology in System Development. s.l. : Wiley, 1995.
35. Beck, Kent. *Extreme Programming Explained*. s.l. : Addison-Wesley, 1997.
36. Darimont, Robert, et al. GRAIL/KAOS: an environment for goal-driven requirements engineering. 1997. Proceedings of the 19th international conference on Software engineering. pp. 612-613.
37. Yu, E.S.K. Towards modelling and reasoning support for early-phase requirements engineering. 1997. Proceedings of ISRE '97: 3rd IEEE International Symposium on Requirements Engineering. pp. 226-235.
38. Kang, K., Cohen, S., Hess, J., Nowak, W., & Peterson, S. *Feature-Oriented Domain Analysis (FODA) Feasibility Study*. Pittsburg: Software Engineering Institute, Carnegie Mellon University., 1990.
39. Spivey, JM and Abrial, J.R. *The Z notation*. s.l. : Prentice Hall, 1992.
40. Pokharel, P. and Vaidya, P. Bahrain. A Study of User Story in Practice.: s.n., 2020. 2020 International Conference on Data Analytics for Business and Industry: Way Towards a Sustainable Economy (ICDABI). pp. 1-5.
41. Ronanki, Krishna, Berger, Christian and Horkoff, Jennifer. Investigating ChatGPT's potential to assist in requirements elicitation processes. s.l. : IEEE, 2023. 49th EuroMicro conference on software engineering and advanced applications (SEAA).
42. Rahman, Tajmilur and Zh, Yuecai. Automated user story generation with test case specification using large language model. 2024.
43. Sami, M. A., Waseem, M., Zhang, Z., Rasheed, Z., Systä, K., & Abrahamsson, P. *AI based Multiagent Approach for Requirements Elicitation and Analysis*. 2024.
44. Hymel, Cory and Johnson, Hiroe. Analysis of LLMs vs Human Experts in Requirements Engineering. 2025.