Improving Agentic AI Opportunity Identification Through GORE Driven Prompt Engineering

Edson Andrade de Moraes
Informatics Department, PUC-Rio
Rio De Janeiro, Brazil
emoraes@inf.puc-rio.br

Marcos Kalinowski
Informatics Department, PUC-Rio
Rio de Janeiro, Brazil
kalinowski@inf.puc-rio.br

Abstract

Agentic AI has emerged as a promising strategy for modernizing legacy software systems; however, systematic approaches for identifying Agentic AI implementation opportunities from existing requirements documentation remain largely unexplored. This paper presents a small-scale industrial study that investigates whether the i* framework can be used as an intermediate Prompt Engineering artifact to improve the identification of Agentic AI opportunities in legacy enterprise systems. The study analyzes the requirements specification of an ERP-based maritime logistics solution supporting fiscal document processing, commercial operations, data migration, system integration, and regulatory compliance activities. Two prompt-based approaches were compared: (i) direct analysis of requirements documentation and (ii) a two-step process using Strategic Dependency and Strategic Rationale models derived from i*. Results indicate that the use of i* did not increase the number or coverage of identified opportunities. However, it produced a more specialized and organizationally aligned multi-agent architecture, improving the connection between proposed agents, organizational objectives, business dependencies, and quality concerns. The findings suggest that i* can serve as an effective intermediary artifact for Prompt Engineering, enabling richer architectural reasoning and providing stronger organizational grounding for Agentic AI solutions in software modernization initiatives.

Keywords

Requirements Engineering, Agentic AI

1 Introduction

Agentic AIs constitute a qualitative leap in the development of artificial intelligence, defined by their capability to set complex goals in a changing and uncontrolled situation and to pursue them through autonomously managing their resources [2]. However, a significant technical limitation lies in integrating Agentic AI with legacy systems that were not originally designed to support it, making implementation in many industries both complex and expensive [17]. Even with its associated difficulties, there is evidence of an industry movement in this direction. For example, Sarferaz [19] proposes a systematic approach for developing Agentic AI business applications within ERP environments, where AI agents combine prompt-based reasoning with the execution of business capabilities exposed as tools over the enterprise platform.

This process, of evolving a legacy system or elements of the system, when conventionally evolutionary practices, such as maintenance and evolution, can no longer achieve the desired system properties, is commonly referred to in the literature as software modernization [20]. In a recent review of the literature on this issue [5], the authors identify a broad range of modernization strategies, from cloudification and architecture redesign to database modernization and digital transformation. However, they do not identify any modernization strategies specifically related to the infusion of Agentic AI in legacy applications. Moreover, none of the identified strategies uses existing requirements documentation as a starting point for the modernization process. The authors highlight scenarios involving legacy systems that lack architectural documentation and argue that tools capable of recovering software architectures through direct code inspection would be highly beneficial, as discussed in [23]. But it is important to mention that this is not always the scenario.

In[15], an exploratory study on documentation practices and knowledge sharing in large-scale ES projects, it is acknowledged that the common reasons for documentation creation and maintenance are legal and contractual mandates across different industries. The pharmaceutical, finance, and health sectors, for example, have high compliance obligations. In these sectors there is a discipline to maintain processes, practices and documentation around it.

Once there is quality requirements documentation available, due to the pervasiveness of AI tools, a natural option is the use of Gen AI to explore and automate the proposition of possible implementation approaches. In [12], a review of Generative AI for software architecture, the authors identify a significant adoption of GenAI for architectural decision support and architectural reconstruction. Requirement-to-code and Architecture-to-code are the Software Architecture Life Cycle (SALC) phases where LLMs are mostly applied, and have been applied mostly to the initial stages of SALC. Among the most frequent challenges are model precision, hallucinations, ethical aspects, privacy issues, lack of architecture-specific datasets, and the absence of sound evaluation frameworks.

The main contribution of this work is the investigation of the i* framework as an intermediate Prompt Engineering artifact for the identification of Agentic AI opportunities in legacy systems. Through a comparative industrial study involving two prompt-based approaches, we evaluate whether intentional and organizational knowledge captured by i* models can improve the resulting Agentic AI architecture. The study contributes empirical evidence regarding the effects of i* on traceability, architectural quality, and agent identification, while also providing an initial requirements-driven methodology for supporting Agentic AI-based software modernization initiatives.

In this study, an Agentic AI implementation opportunity is defined as a capability, decision point, or business activity described in the requirements specification that could be delegated to an autonomous or semi-autonomous AI agent. Rather than proposing

complete modernization solutions, the goal is to identify architectural candidates for Agentic AI adoption directly from the requirements documentation. This aligns with the early phases of the Software Architecture Life Cycle (SALC), where requirements and architectural knowledge are analyzed to support system evolution and architecture-related decision making.

2 Background

The choice for the i* modeling framework [32] was mainly due to a perceived necessity of a shift-left in the analysis of the requirements documentation. Like in [30], the understanding of the organizational context and rationales (the "Whys") that lead up to systems requirements. We believe this approach would improve the chances of a more accurate identification of Agentic AI opportunities in existing requirements documentation. It is important to note that the use of GORE(Goal Oriented Requirements) is not a novelty.Ahmad et al. [3] analyzed a total of 12 studies that used modeling languages or requirements notation to present requirements. The most popular modeling notations and languages among the studies were UML and GORE. According to the authors, some favored GORE as it had better support for NFR's and business rules. In [7], the authors demonstrate through an empirical study that GORE, particularly through the KAOS approach, is a promising method for teaching requirements engineering for AI-based systems. And also in [31], the author proposes 9 pivots for a re-conceptualization of requirements engineering, based on the i* model, and each of them elaborates and extends the agent-oriented modeling paradigm to encompass a richer ontology and a broader vision of RE.

The agent taxonomy adopted in this study was not derived from a single established classification scheme. Instead, it was synthesized from recurring role patterns reported in recent AI agent literature [28] and industrial Agentic AI frameworks [6]. Classical Multi-Agent Systems (MAS) research [27] emphasizes coordination, specialization, and delegation among autonomous agents, whereas contemporary Agentic AI architectures frequently distinguish planning, reasoning, tool utilization, execution, and review as separate responsibilities. Based on these recurring patterns, we adopted six agent categories—Coordinator, Planner, Reasoning, Tool, Executor, and Reviewer Agents—to provide a pragmatic and self-explanatory classification that supports separation of concerns and facilitates architectural analysis in enterprise modernization scenarios.

3 Study Design

According to classifications of empirical software engineering research [26], this study can be characterized as a small-scale study because it investigates a limited number of treatments in a single industrial setting. Its goal is not statistical generalization but rather the generation of preliminary comparative evidence and practical insights regarding the strengths and limitations of the evaluated approaches, providing a foundation for future replications and larger-scale industrial investigations.

By comparing the results produced by two prompt-based approaches, we seek to investigate whether the inclusion of i* models as intermediary artifacts improves the identification of Agentic AI opportunities in legacy systems. The study was guided by the following research question (RQ) and associated sub-question.

RQ1: Does the use of the i* framework as an intermediate Prompt Engineering artifact improve the identification of Agentic AI implementation opportunities in legacy system requirements documents?

RQ1.1: Does the i* model increase the number and coverage of identified Agentic AI opportunities?

RQ1.2: Does the i* model improve traceability between requirements, organizational goals, and the proposed agents?

RQ1.3 Does the i* model improve the architectural quality of multi-agent solutions generated by Large Language Models (LLMs)?

RQ1.4 Which elements of the i* model (Goals, Tasks, Resources, Softgoals, and Dependencies) contribute the most to the discovery and identification of agents?

3.1 Study Objects and Data Collection

The study was conducted within a state-owned company operating in the energy sector in South America. A complete requirements specification retrieved from the organization's corporate requirements documentation repository was selected as the object of analysis. The selected documentation pertains to a ship logistics execution solution implemented on the company's enterprise resource planning (ERP) platform. The specification was produced using the organization's standardized requirements templates, which are largely derived from the Rational Unified Process (RUP) use-case documentation model and extended with organization-specific customizations. These templates comprise use-case identification and description, detailed flows of events, pre-conditions and post-/conditions, «include» and «extend» relationships, non-functional requirements, and business rules. The data analyzed by the proposed prompts was limited exclusively to the textual content of the specification documents.

During the analysis of the requirements documentation, a recurring process-related issue was identified. According to the company's requirements documentation standards and audit guidelines, only the specific intervention being implemented was required to be explicitly documented. Consequently, the application—originally developed nearly two decades ago and subjected to more than 24 evolutionary maintenance cycles throughout its lifetime—accumulated documentation that primarily reflected incremental changes rather than a comprehensive view of the system requirements.

Although the overall quality of the documentation was generally satisfactory, successive modifications to the requirements templates and occasional changes to the versioning scheme frequently resulted in the reinitialization of documentation artifacts. In such cases, newly created documents captured only the requirements introduced during the corresponding intervention, typically expressed as new use cases and business rules, while previously existing requirements were referenced only indirectly through earlier document versions. This practice led to fragmented documentation and incomplete historical traceability of requirements.

Given the recurrent nature of this issue, we selected for analysis the most recent document that also provided the most comprehensive description of the system requirements. This document was produced in 2012 and, despite approximately 15 subsequent interventions being recorded within the same documentation set over the following years, the majority of its documented requirements

remained valid at the time of the study. Although it would have been possible to employ an LLM to generate a consolidated version of all available documents, we deliberately avoided this approach. Introducing LLM-generated consolidation could have created a confounding factor, making it difficult to isolate and accurately assess the effects of the prompt identification approaches investigated in this study on the final results.

The requirements document has a total of 13 use cases with 7 explicit actors. If we consider specializations of the actors we have the following: one General Business User (Commercial), three Specialized Roles, one Organizational Area Actor, one Satellite System Actor and one Satellite System–ERP Interface System actor. There are 16 business rules with a total of 6 of them which are reused.

The use cases are organized as follows. For each use case, the identifier (UC), the use case name, the number of associated actors, and the number of business rules are presented. The names of some of the elements were changed to better understand the solution and also to preserve company information.

- **UC1 – Reverse Invoice via ERP Transaction**: 1 actor and 1 business rule.
- **UC2 – View Scale Invoices**: 2 actors and 1 business rule.
- **UC3 – Issue Invoice**: 2 actors and 7 business rules.
- **UC4 – Issue Provisional Invoice**: 1 actor and 2 business rules.
- **UC5 – Issue Definitive Invoice**: 1 actor and 2 business rules.
- **UC6 – Issue Incoming Invoice**: 1 actor and 6 business rules.
- **UC7 – Reverse Incomplete Actions**: 1 actor and 1 business rule.
- **UC8 – Enter Measurement Parameters**: 1 actor and 1 business rule.
- **UC9 – Migrate Satellite System Data**: 1 actor and 2 business rules.
- **UC10 – Perform Commercial Release**: 1 actor and 1 business rule.
- **UC11 – Perform Advance Receipt**: 1 actor and 1 business rule.
- **UC12 – Perform Manual Shore Confirmation**: 1 actor and 1 business rule.
- **UC13 – Execute Satellite-ERP Interface**: 1 actor and 1 business rule.

The prompts were written in Portuguese to match the documentation language and avoid potential analysis bias. For the english version of the prompt, please refer to the Artifacts section.

3.2 Study Execution

As a basis for constructing our prompts, we used the prompt engineering guideline mapping proposed by Ronanki et al. [18]. This mapping consolidates recommendations identified through a systematic literature review and validated through interviews with Requirements Engineering experts.Based on this mapping, we selected a subset of the guidelines associated with the analysis phase and incorporated them into the prompt design adopted in this study

Most of the prompt was handwritten, except for the part which creates the Strategic Dependency and the Strategic Rationale models based on the use case documents. This was the adaptation of direct prompt to MS Copilot, which asked suggest a prompt to create the the two i* models from a use case document.

We selected a subset of the prompt engineering design patterns suggested for the analysis phase as the basis for the construction of our prompts. All prompts were performed by the Microsoft 365 Copilot (M365 Copilot), based on the GPT-5 chat model, assistant. The chosen patterns are the following:

- Self-consistency boosts the performance of chain-of-thought prompting [25]: rationale augmented ensembles achieve more accurate and interpretable results than existing prompting approaches—including standard prompting without rationales and rationale-based chain-of-thought prompting—while simultaneously improving interpretability of model predictions through the associated rationales.
- Tree of thought(ToT) [29]: allows LLMs to perform deliberate decision making by considering multiple different reasoning paths and self-evaluating choices to decide the next course of action

4 Study Results and Discussion

4.1 RQ1.1: Does the i* model increase the number and coverage of identified Agentic AI opportunities?

Table 1 summarizes the identified agents per category by the two prompts. The first indicates the agents per category identified only referencing the use cases in the document(Use Case Only) and the second one (Use Case + i*) the agents identified per category using the use cases and the intermediate i* models.

Within the context of this study, the use of i* models did not appear to affect the number or coverage of identified Agentic AI implementation opportunities. In fact both solutions covered all the necessary requirements for the solution. No use case or business rule was left without coverages in both. But it changed considerably the proposed Agent Architecture which will be treated in the next RQs.

4.2 RQ1.2: Does the i* model improve traceability between requirements, organizational goals, and the proposed agents?

The results provide evidence that the use of the i* framework improved traceability between requirements, organizational goals, business processes, and the proposed Agentic AI architecture. Unlike a purely use-case-driven analysis, the i* models enabled agents to be traced not only to functional requirements but also to the organizational motivations that justify their existence. In this study, all identified agents were derived from elements captured in the Strategic Dependency (SD) and Strategic Rationale (SR) models, which were constructed from the 13 use cases, 16 business rules, and 7 actors described in the specification.

The Strategic Dependency model contributed by revealing organizational dependencies among actors and systems. Dependencies

Table 1: Comparison of Identified Agents Across Experiments

Agent Category	Use Case Only	Use Case + i*
Coordinator Agent	Operations Coordinator Agent	Logistics and Fiscal Coordinator Agent
Planner Agent	Fiscal Planning Agent	Operational Planning Agent
Reasoning Agent	Fiscal Reasoning Agent Cost Center Validation Agent	Fiscal Reasoning Agent Reversal Eligibility Reasoning Agent Cost Center Validation Agent
Tool Agent	Satellite System Integration Agent ERP Integration Agent Invoice Monitoring Agent	Satellite System Integration Agent
Executor Agent	Fiscal Execution Agent Migration Execution Agent	ERP Fiscal Execution Agent Reversal Execution Agent Migration Execution Agent
Reviewer Agent	Fiscal Compliance Review Agent Integration Review Agent Exception Analysis Agent	Fiscal Compliance Review Agent Data Quality Review Agent Audit and Traceability Review Agent

related to invoice issuance, invoice reversal, data migration, cost-center validation, Invoice authorization, and compliance verification exposed coordination and information-exchange requirements that translated directly into agent responsibilities. For example, dependencies associated with invoice processing led to the identification of the Fiscal Planning Agent and Fiscal Execution Agent, while dependencies involving Satellite System, ERP, and integration interfaces motivated the creation of specialized integration agents.

The Strategic Rationale model strengthened the identification process by linking agents to actor objectives and organizational concerns. Goals related to invoice issuance, fiscal reversals, logistics data migration, compliance, and cost-center validation became candidate agent objectives. In addition, softgoals such as traceability, reliability, auditability, data quality, fiscal compliance, and integration consistency justified the introduction of reviewer-oriented agents, such as the Fiscal Compliance Review Agent, Integration Review Agent, and Exception Analysis Agent.

A comparison between i* elements and the identified agents revealed a clear pattern: goals mainly contributed to Planner and Executor agents, dependencies to Coordinator and Integration agents, and softgoals to Reviewer agents. As a result, the i* analysis provided a stronger rationale for agent identification than a purely functional decomposition based solely on use cases.

Overall, the main contribution of the i* framework was not the discovery of additional agent opportunities, but the improvement of traceability and architectural justification. The resulting architecture established explicit links between agents, organizational goals, business dependencies, resources, and quality concerns, producing a more explainable and organizationally aligned Agentic AI solution.

4.3 RQ1.3 Does the i* model improve the architectural quality of multi-agent solutions generated by LLMs?

In this study, architectural quality was assessed in terms of specialization, traceability, governance support, and alignment with organizational goals. The main architectural difference between the two multi agent architecture proposed solutions lies in the level of specialization adopted for process execution and governance responsibilities. The architecture proposed by the Use Case Only prompt is primarily oriented toward system integration and operational automation, introducing dedicated agents for ERP integration, Satellite System integration, and Invoice monitoring. This design emphasizes interoperability among enterprise systems and the automation of logistics and fiscal workflows. As a result, integration concerns are explicitly represented through specialized Tool Agents responsible for communication, data exchange, and synchronization across heterogeneous systems.

In contrast, the i* enhanced architecture exhibits a stronger focus on governance, compliance, and data quality management. Rather than allocating multiple agents to system integration activities, it introduces additional agents dedicated to reasoning about reversal eligibility, executing reversal operations, reviewing data quality, and ensuring auditability and traceability. This approach decomposes responsibilities into more granular and specialized agents, thereby increasing the level of oversight and control over business processes.

Another relevant difference concerns the treatment of execution activities. In the architecture proposed in the Use Case Only prompt, a single Fiscal Execution Agent concentrates most fiscal execution responsibilities. Conversely, the alternative architecture separates these responsibilities into an ERP Fiscal Execution Agent and a Reversal Execution Agent, resulting in a finer-grained execution layer. Similarly, while the use case only employs a generic Exception Analysis Agent to handle exceptional situations, the i* enhanced architecture distributes these concerns across dedicated Data Quality Review and Audit and Traceability Review agents.

Within the context of this study, the i*-enhanced approach produced a multi-agent architecture that exhibited stronger alignment with organizational goals and governance concerns. We consider this the most significant finding of the study, as it suggests that the proposed shift-left approach to requirements analysis was successful in enhancing the understanding of the underlying rationale behind system requirements [30]. Rather than merely reproducing

Table 2: 2 Most Significant i* Elements Contributing to Agent Identification

Proposed Agent	Key i* Elements
Operations Coordinator Agent	Dependencies, Goals
Fiscal Planning Agent	Goals, Tasks
Fiscal Reasoning Agent	Goals, Dependencies
Cost Center Validation Agent	Goals, Resources
Satellite System Integration Agent	Dependencies, Resources
ERP Integration Agent	Dependencies, Resources
Invoice Monitoring Agent	Dependencies, Softgoals
Fiscal Execution Agent	Tasks, Goals
Migration Execution Agent	Tasks, Dependencies
Fiscal Compliance Review Agent	Softgoals, Goals
Integration Review Agent	Softgoals, Dependencies
Exception Analysis Agent	Softgoals, Tasks

the existing legacy system architecture using Agentic AI technologies, the resulting architecture exhibited a stronger alignment with organizational goals, business motivations, and governance concerns.

4.4 RQ1.4 Which elements of the i* model (Goals, Tasks, Resources, Softgoals, and Dependencies) contribute the most to the discovery and identification of agents?

Among the i* elements, Goals and Dependencies appeared to be the most influential elements to the discovery and identification of agents, as seen in Table 2. Goals revealed the intentional objectives that required autonomous behavior, leading to the identification of planning, reasoning, and execution agents. Dependencies exposed coordination, information exchange, and responsibility relationships among actors and systems, driving the identification of coordinator, integration, and monitoring agents. Tasks and Resources mainly refined agent responsibilities after the agents had already been identified, while Softgoals were particularly important for discovering reviewer agents focused on compliance, auditability, reliability, and traceability. Therefore, Goals provided the primary source for defining agent purposes, whereas Dependencies provided the primary source for defining agent interactions and boundaries, making these two i* elements the most influential in the overall agent identification process.

4.5 RQ1: Does the use of the i* framework as an intermediate Prompt Engineering artifact improve the identification of Agentic AI implementation opportunities in legacy system requirements documents?

The findings suggest that the use of the i* framework as an intermediate Prompt Engineering artifact can improve the agent identification process by strengthening traceability and organizational alignment. However, this improvement did not result from a higher number of discovered agentic opportunities or a broader coverage of the requirements domain. The use case analysis alone was already able to identify most of the relevant operational, integration, and automation opportunities.

The main contribution of the i* models was observed in the quality of the resulting multi-agent architecture. By explicitly modeling actors, goals, dependencies, resources, tasks, and softgoals, the i* framework provided additional organizational context that was not directly apparent in the use cases. This context enabled a stronger alignment between the proposed agents and the underlying business objectives, stakeholder intentions, and organizational dependencies.

5 Limitations and Future Directions

One important threat to validity int his study, concerns the attribution of the observed improvements to the i* framework itself. Part of the observed benefits may derive from the intermediate modeling step itself rather than the specific notation employed. An interesting direction for future work is the evaluation of alternative Goal-Oriented Requirements Engineering (GORE) approaches, such as Tropos [9] and KAOS [11], as intermediary models for identifying Agentic AI implementation opportunities. Likewise, other requirements engineering artifacts, including User Stories [8] and Behavior-Driven Development (BDD) specifications [16], should be investigated. Such studies could help determine whether the observed architectural improvements arise from specific characteristics of i* and GORE models or from a broader reasoning process induced by the intermediate modeling step.

Another limitation of the proposed approach is its exclusive reliance on requirements specifications as the sole input artifact. Although our study showed that high-quality requirements documentation can support the generation of satisfactory Agentic AI architectural proposals, requirements alone may not fully capture the operational reality of long-lived legacy systems. Even in organizations with mature documentation practices, incremental updates, evolving templates, and partial versioning can result in incomplete representations of system behavior. Furthermore, requirements documents often omit tacit [21] and implicit knowledge [13]. It is also important to note that our evaluation was restricted to use case models, which limits the generalizability of the results and motivates further studies using alternative forms of requirements documentation, such as user stories [8] and vision documents [14].

These limitations motivate the use of additional artifacts as complementary inputs. Even when architectural documentation is available, architectural drift may create discrepancies between documented architectures and their actual implementations [4, 24]. Source code is therefore a particularly promising artifact, as it captures concrete system behavior and dependencies. Other structured artifacts, including legislation, regulatory texts, and process standards, also deserve investigation. Prior studies have already demonstrated the feasibility of extracting requirements from regulatory documents [1], suggesting that such artifacts may provide valuable organizational, compliance, and business context for Agentic AI opportunity identification.

Finally, future work should investigate alternative mechanisms for processing and combining these artifacts. While prompt engineering remains an active research area with several design patterns

yet to be explored [18, 22], other interaction paradigms for applying large language models in software engineering have also emerged. In particular, retrieval-augmented generation (RAG) approaches have been employed for requirements-related activities, including requirements elicitation [10]. Adapting similar mechanisms to support software modernization represents an interesting direction for future investigation. Additionally, due to the scope of this study, the experiment was conducted using a single LLM. Evaluating the influence of different models on the quality and consistency of the generated architectures constitutes another important avenue for future research.

Artifact Availability

The prompt used in this study is publicly available at https://doi.org/10.5281/zenodo.21301114 and is released under the MIT License.

Acknowledgements

We gratefully acknowledge the support of CAPES (Finance Code 001 and PROEX), CNPq (Grants 312275/2023-4 and 420191/2025-9), FAPERJ (Grant E-26/204.256/2024), and Instituto Kunumi for their generous support.

References

[1] Sallam Abualhaija, Marcello Ceci, Nicolas Sannier, Domenico Bianculli, Salomé Lannier, Martina Siclari, Olivier Voordeckers, and Stanisław Tosza. 2025. LLM-assisted Extraction of Regulatory Requirements: A Case Study on the GDPR. In *2025 IEEE 33rd International Requirements Engineering Conference (RE)*. IEEE, Valencia, Spain, 142–154. doi:10.1109/RE63999.2025.00023

[2] Deepak Bhaskar Acharya, Karthigeyan Kuppan, and B. Divya. 2025. Agentic AI: Autonomous Intelligence for Complex Goals—A Comprehensive Survey. *IEEE Access* 13 (2025), 18912–18936. doi:10.1109/ACCESS.2025.3532853

[3] Khlood Ahmad, Mohamed Abdelrazek, Chetan Arora, Muneera Bano, and John Grundy. 2023. Requirements engineering for artificial intelligence systems: A systematic mapping study. *Information and Software Technology* 158 (June 2023), 107176. doi:10.1016/j.infsof.2023.107176

[4] Nour Ali, Sean Baker, Ross O'Crowley, Sebastian Herold, and Jim Buckley. 2018. Architecture consistency: State of the practice, challenges and requirements. *Empir Software Eng* 23, 1 (Feb. 2018), 224–258. doi:10.1007/s10664-017-9515-3

[5] Wesley K. G. Assunção, Luciano Marchezan, Lawrence Arkoh, Alexander Egyed, and Rudolf Ramler. 2025. Contemporary Software Modernization: Strategies, Driving Forces, and Research Opportunities. *ACM Trans. Softw. Eng. Methodol.* 34, 5 (June 2025), 1–35. doi:10.1145/3708527

[6] AWS. 2026. Agentic AI patterns and workflows on AWS - AWS Prescriptive Guidance. Retrieved 2026-07-10 from https://docs.aws.amazon.com/prescriptive-guidance/latest/agentic-ai-patterns/introduction.html

[7] Beatriz Braga Batista, Márcia Sampaio Lima, and Tayana Uchôa Conte. 2024. Teaching Requirements Engineering for AI: A Goal-Oriented Approach in Software Engineering Courses. In *Proceedings of the XXIII Brazilian Symposium on Software Quality*. ACM, Salvador Bahia Brazil, 613–623. doi:10.1145/3701625.3701686

[8] Kent Beck. 2003. *Extreme Programming explained: embrace Change* (9. print ed.). Addison Wesley, Boston, Mass. Munich.

[9] Paolo Bresciani, Anna Perini, Paolo Giorgini, Fausto Giunchiglia, and John Mylopoulos. 2004. Tropos: An Agent-Oriented Software Development Methodology. *Autonomous Agents and Multi-Agent Systems* 8, 3 (May 2004), 203–236. doi:10.1023/B:AGNT.0000018806.20944.ef

[10] Hayala Nepomuceno Curto. 2025. Building Software Functional Requirements Lists Using RAG with Distinct LLMs in Multiple Interactions. In *2025 IEEE 33rd International Requirements Engineering Conference (RE)*. IEEE, Valencia, Spain, 612–616. doi:10.1109/RE63999.2025.00077

[11] Anne Dardenne, Axel Van Lamsweerde, and Stephen Fickas. 1993. Goal-directed requirements acquisition. *Science of Computer Programming* 20, 1-2 (April 1993), 3–50. doi:10.1016/0167-6423(93)90021-G

[12] Matteo Esposito, Xiaozhou Li, Sergio Moreschini, Noman Ahmad, Tomas Cerny, Karthik Vaidhyanathan, Valentina Lenarduzzi, and Davide Taibi. 2026. Generative AI for software architecture. Applications, challenges, and future directions. *Journal of Systems and Software* 231 (Jan. 2026), 112607. doi:10.1016/j.jss.2025.112607

[13] E. Hudlicka. 1996. Requirements elicitation with indirect knowledge elicitation techniques: comparison of three methods. In *Proceedings of the Second International Conference on Requirements Engineering*. IEEE Comput. Soc. Press, Colorado Springs, CO, USA, 4–11. doi:10.1109/ICRE.1996.491424

[14] Philippe Kruchten. 2007. *The rational unified process: an introduction* (3. ed., 7. printing ed.). Addison-Wesley, Upper Saddler River, NJ.

[15] Makoto Nakayama, Eli Hustad, and Norma Sutcliffe. 2021. Agility and system documentation in large-scale enterprise system projects: a knowledge management perspective. *Procedia Computer Science* 181 (2021), 386–393. doi:10.1016/j.procs.2021.01.181

[16] Dan North. 2006. Introducing BDD. Retrieved 2026-07-10 from https://dannorth.net/blog/introducing-bdd/

[17] Tayiba Raheem and Gahangir Hossain. 2025. Agentic AI Systems: Opportunities, Challenges, and Trustworthiness. In *2025 IEEE International Conference on Electro Information Technology (eIT)*. IEEE, Valparaiso, IN, USA, 618–624. doi:10.1109/eIT64391.2025.11103638

[18] Krishna Ronanki, Simon Arvidsson, and Johan Axell. 2026. Prompt Engineering Guidelines for Using Large Language Models in Requirements Engineering. In *Software Engineering and Advanced Applications*, Davide Taibi and Darja Smite (Eds.). Vol. 16083. Springer Nature Switzerland, Cham, 245–262. doi:10.1007/978-3-032-04207-1_17 Series Title: Lecture Notes in Computer Science.

[19] Siar Sarferaz. 2025. Implementing Agentic AI Into ERP Software. *IEEE Access* 13 (2025), 178945–178960. doi:10.1109/ACCESS.2025.3621887

[20] Robert C. Seacord, Daniel Plakosh, and Grace A. Lewis. 2003. *Modernizing legacy systems: software technologies, engineering processes, and business practices*. Addison-Wesley, Boston Munich.

[21] A. Stone and P. Sawyer. 2006. Identifying tacit knowledge-based requirements. *IEE Proc., Softw.* 153, 6 (2006), 211. doi:10.1049/ip-sen:20060034

[22] Irdina Wanda Syahputri, Eko K. Budiardjo, and Panca O. Hadi Putra. 2025. Unlocking the Potential of the Prompt Engineering Paradigm in Software Engineering: A Systematic Literature Review. *AI* 6, 9 (Aug. 2025), 206. doi:10.3390/ai6090206

[23] Damian A. Tamburri and Rick Kazman. 2018. General methods for software architecture recovery: a potential approach and its evaluation. *Empir Software Eng* 23, 3 (June 2018), 1457–1489. doi:10.1007/s10664-017-9543-z

[24] Burak Uzun and Bedir Tekinerdogan. 2024. Detecting deviations in the code using architecture view-based drift analysis. *Computer Standards & Interfaces* 87 (Jan. 2024), 103774. doi:10.1016/j.csi.2023.103774

[25] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, and Denny Zhou. 2022. Rationale-Augmented Ensembles in Language Models. doi:10.48550/ARXIV.2207.00747 Version Number: 1.

[26] Claes Wohlin and Austen Rainer. 2022. Is it a case study?—A critical analysis and guidance. *Journal of Systems and Software* 192 (Oct. 2022), 111395. doi:10.1016/j.jss.2022.111395

[27] Michael Wooldridge. 2009. *An introduction to multiagent systems*. John wiley & sons.

[28] Bin Xu. 2026. *AI Agent Systems: Architectures, Applications, and Evaluation*. arXiv:2601.01743 doi:10.48550/ARXIV.2601.01743

[29] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of Thoughts: Deliberate Problem Solving with Large Language Models. In *Advances in Neural Information Processing Systems*, A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (Eds.), Vol. 36. Curran Associates, Inc., 11809–11822. https://proceedings.neurips.cc/paper_files/paper/2023/file/271db9922b8d1f4dd7aaef84ed5ac703-Paper-Conference.pdf

[30] E.S.K. Yu. 1997. Towards modelling and reasoning support for early-phase requirements engineering. In *Proceedings of ISRE '97: 3rd IEEE International Symposium on Requirements Engineering*. IEEE Comput. Soc. Press, Annapolis, MD, USA, 226–235. doi:10.1109/ISRE.1997.566873

[31] Eric Yu. 2024. Agent-Oriented Modeling for the Age of AI: Nine Pivots Toward a Reconceptualization of Requirements Engineering. In *Social Modeling Using the i* Framework*, Xavier Franch, Julio Cesar Sampaio Do Prado Leite, Gunter Mussbacher, John Mylopoulos, and Anna Perini (Eds.). Springer Nature Switzerland, Cham, 207–227. doi:10.1007/978-3-031-72107-6_13

[32] Eric S. K. Yu (Ed.). 2011. *Social modeling for requirements engineering*. MIT Press, Cambridge, Mass.