Trust and Hallucinations: A Study of 39 Experts on AI-Assisted Requirements Reverse Engineering

Abdullah A H Alzahrani
Department of Computers, Engineering and Computing College at Alqunfuda,
Umm Al Qura University Makkah, Saudi Arabia

***Abstract*—With regard to the evolution of software systems, the process is hindered by the poor state of documentation, as software systems continuously evolve, which thereafter increases the maintenance costs to around 90% of development lifecycle spending. In addition, although the extraction of embedded business logic through the reverse engineering of requirements is essential, a gap in meaning remains between the source code and the high-level objectives, which means a need for addressing this issue. Therefore, currently, many artificial intelligence tools are in place for such actions. This research evaluates the performance of specialized Retrieval-Augmented Generation (RAG), general-purpose large language models, and hybrid static AI systems by focusing on the expert observations of practitioners within industrial environments. To achieve this, the study gathers data to measure hallucination rates and the accuracy of business rule recovery based on the actual professional experience of those managing legacy code. In particular, these experts used EPAM ART, GitHub Copilot, and IBM ADDI to provide percentage-based error estimates and rate rule identification on a standard scale. Ultimately, this empirical approach ensures that the research questions are addressed through the practical insights and lived experiences of professionals. In this research, a study of perspectives of 39 senior professionals observed that, while general models are successful at abstracting meaning with a score of 4.05 out of 5, a shortfall in traceability is retained. Furthermore, it was discovered that hybrid tools such as IBM ADDI allow for superior formal mapping with a score of 4.23 out of 5, although a struggle in verification is produced because high rates of incorrect data generation or hallucination exceeding 20% were reported by 66.7% of the participants. In light of these findings, this research proposes a strategy of multiple tool coordination in order to make the evolution of software systems feasible over long periods.**

***Keywords*—Software engineering (SE); requirements engineering (RE); requirements reverse engineering (RRE); large language models (LLMs); natural language processing (NLP)**

I. Introduction

A. Context and Problem Statement

Regarding the operation of modern enterprises, legacy software systems are viewed as the functional base, although the fading of documentation is frequently observed as original requirements no longer align with the evolving source code [1-6]. In addition, this state results in obsolete documentation that affects the maintainability of systems and increases the risk of architectural erosion during modernization efforts. Furthermore, as a shift toward microservices and cloud native architectures is accepted by organizations, the reverse engineering of requirements is required so that business rules embedded in undocumented implementations in the software are recovered. As a result, the difficulty of translating low level programming constructs into human readable requirements is encountered as a gap of semantics, additionally, while manual processes are considered economically not feasible due to costs reaching 90% of lifecycle expenses, traditional rule based automation is hindered by the lack of cognitive flexibility for complex logic [7-11].

Regarding the potential for modern artificial intelligence and large language models to offer a useful approach to bridge this gap, the deployment of such tools is hindered by risks as these models are prone to the generation of nonsense requirements or hallucinations that contradict the actual source code logic [12-18]. In addition, a shortage in traceability is maintained, where high-level summaries are provided without a detailed mapping to the specific lines of code from which they were derived. As a result, this absence of mapping is found to make the verification process more difficult. Due to the influence of these factors, a struggle in verification is produced where the role of the engineer is required to change from an author to an oversight manager who is responsible for validating large generated output so that system functionality and integrity are ensured [19-20].

B. Motivation

In order to allow the integration of artificial intelligence into the reverse engineering of requirements, the process is imposed by a demand for economic improvement throughout the software lifecycle because maintenance and evolution are found to account for 60% to 90% of the total cost of software development [1, 21-26]. In addition, a large majority of these costs are produced by the cognitive effort that is required to understand undocumented legacy code, while it is also observed that artificial intelligence acts as a safeguard against the loss of institutional knowledge where business logic is removed as senior engineers depart. Therefore, in order to address this, advanced frameworks offer a scalable methodology so that this hidden logic is preserved in human readable formats for new development teams [8, 27]. Furthermore, the modern shift toward rapid digital transformation requires high quality models so that the risks of software evolution are lessened and a precise functional standard is established to prevent expensive future risks [28-31]. Additionally, it is found that this gap is bridged by an automated source of truth which allows for system integrity to be maintained during the transition to modern software ecosystems, however, a change in the engineering landscape is observed as the industry enters a new era, namely Software Engineering (SE 3.0) paradigm, of agentic software engineering

where artificial intelligence acts as an active partner. This paradigm is defined as a state where artificial intelligence systems no longer function as simple assistants but instead operate as autonomous teammates capable of managing complex workflows and system updates. Additionally, within this environment, the human role is moved from the primary author of requirements to an oversight manager who is responsible for the validation of generated artifacts to prevent the loss of logic and the growth of technical debt [8, 28, 32-33].

C. Research Questions

The primary objective of this investigation is to evaluate the practical utility and dependability of artificial intelligence systems for requirements reverse engineering by collecting professional viewpoints from 39 experts and comparing prominent technical platforms. To meet this objective, several inquiries are pursued by the study. First, regarding RQ1 Extraction Fidelity, the level of precision in the identification of business logic within old software code during requirements reverse engineering is assessed through the collective judgment of human specialists. Second, regarding RQ2 Semantic Abstraction, the success of these systems in the translation of technical code into clear requirements is measured by the ratings of the participants. Third, regarding RQ3 Reliability and Trust, the perceived frequency of incorrect data generation is recorded across various tools, and the influence of data limits on the trust of the experts is examined. Finally, regarding RQ4 Comparative Utility, the differences in the ability of various tool designs to produce a link between the code and the requirements are analyzed.

D. Research Contributions

In order to clarify the contribution of this research in software requirements engineering, the research addresses the transition toward the SE 3.0 paradigm through several key findings. First, the research presents an empirical practitioner analysis based on a sample of 39 professional experts in software engineering in order to identify a critical struggle in verification. Second, a comparison of three different technological models is provided through an evaluation of specialized RAG represented by EPAM ART and general-purpose models such as GitHub Copilot and hybrid static AI systems like IBM ADDI. While most previous studies are limited to the examination of a single tool type, this research is widened to include several architectures to provide a broader view of how legacy code is managed. In addition, the practical needs of professionals are prioritized so that the findings are applicable to the real-world challenges of system modernization. Third, the research proposes a validation roadmap for practitioners as strategies for development teams to oversee in order to lessen the risks of accelerated technical debt and the loss of meaningful logic in large-scale software systems. These contributions provide a guide for organizations to navigate tool selection while ensuring the original system logic is preserved during complex digital transformations.

E. Paper Outline

In order to outline this research study's structure, the study is organized into five subsequent sections: Section II provides the theoretical foundation and reviews literature on documentation decline or loss, the gap in meaning, and the three primary failure modes. Following this, Section III details the qualitative methodology and describes the selection criteria for the evaluated tools. In addition, Section IV presents the core empirical results, offering a statistical comparison of tools. Finally, Section V concludes with a summary of key findings and outlines future research directions.

II. Background and Related Work

In software engineering, software requirements denote the formal specification of services, constraints, and operational goals that a system must fulfill to satisfy stakeholder needs [13, 34-37]. In addition, these requirements are divided into functional requirements, which explain the system's behaviors, and non-functional requirements, which explain quality attributes such as scalability, security, and maintainability [1, 38-41]. Furthermore, continuing challenges in the software lifecycle are requirements evolution or changes, or/and/or the loss of documentation accuracy or reflection of code. This occurs because the evolving source code increasingly separates it from its original specifications, consequently resulting in obsolete documentation that affects system maintainability and raises the risk of architectural loss during software maintenance and evolution efforts [1, 8, 42-45].

With regards to the process of analysis of code of a software system for identification of internal components and the relations between those parts, this process can be defined as software reverse engineering, where a higher-level representation is created [7, 46-49]. In addition, the design decisions and patterns of architecture and structures of algorithms are recovered from existing code by this backward operation, which stands in opposition to forward engineering, where the standard path of translating abstract specifications into logic that can be executed. Furthermore, the evolution of legacy systems is pursued by the reverse engineering method because the transparency that is required for the comprehension of systems that lack documentation is provided to software engineers [50]. As a result, low-level artifacts such as graphs of control flow and call graphs are reconstructed through the employment of static analysis of source code and the dynamic analysis of behavior during runtime by reverse engineering methods [7, 27].

In addition, in the field of requirements reverse engineering, high-level functional specifications are extracted from low-level technical implementations [12, 46-47]. Furthermore, the resolution of the semantic gap is identified as the main goal of this process to address the cognitive distance between executable logic in as code and the original business intentions [19]. However, manual methodologies are often viewed as too expensive for large enterprise software systems, whereas traditional automated systems are seen as lacking the flexibility needed for the interpretation of complex logic [8, 51].

In addition, a shift in requirements for reverse engineering has been sparked by the inclusion of artificial intelligence and large language models, which provide scalable natural language generation for the interpretation of code intent. Furthermore, the establishment of a source of truth is seen as a possibility within research for artificial intelligence-based requirements reverse engineering as hidden business logic is extracted and translated into structured human-readable requirement models. As a result,

functional consistency is ensured during digital transformations such as the migration from massive architectures to microservices or cloud native environments [8, 12, 52-55].

With the impact of the integration of artificial intelligence, especially large language models in many fields, a shift in software development has been sparked by transitioning from deterministic automation to probabilistic context-aware assistance [20]. Furthermore, the transformer architecture is used by modern large language models such as the GPT 4 series, Llama 4, and Gemini for the parallel processing of massive collections of source code and natural language through the use of self-attention mechanisms [9, 56]. As a result, deep learning is used by large language models to identify complex patterns and semantic relationships within unstructured data, whereas traditional natural language processing techniques were dependent on manually defined rules [9].

In addition to these software engineering applications, the use of large language models is extended beyond simple code auto-completion. Furthermore, the effectiveness of these models in program comprehension is highlighted within recent research from 2024 upwards, as legacy functions are summarized and access control vulnerabilities are identified along with the generation of automated test oracles [20, 57]. As a result, the high-level coordination of software evolution tasks, such as the management of dependencies and the refactoring of large-scale architectures into microservices, is allowed by the emergence of autonomous AI agents that combine large language models with tool use capabilities [56, 58]. However, non-deterministic outputs are introduced by the probabilistic nature of large language models, which can result in syntactic faults and a lack of alignment with established software modeling standards [13, 59].

A. Critical Challenges in AI-Supported Requirements Reverse Engineering

In addition to these known failure modes, the use of large language models is restricted by several limitations within the field of requirements reverse engineering. Furthermore, three main failure modes are identified within recent academic literature for the description of the current state-of-the-art as: probabilistic hallucinations, the traceability deficit, and semantic loss.

1) Probabilistic hallucinations and factual inconsistency: The industrial adoption of artificial intelligence for requirements reverse engineering is hindered by the occurrence of hallucinations where syntactically correct but erroneous or non-existent business rules are generated as recovered requirements [20, 60]. Furthermore, inconsistencies are frequently produced where the generated requirement contradicts the actual source code logic because large language models operate on a probabilistic next token prediction system rather than a deterministic logic engine [56]. As a result, the common nature of these errors within legacy systems involving older libraries or restricted domain-specific languages is indicated by recent empirical studies where accurate context cannot be derived due to a lack of sufficient training data [13, 60].

2) The traceability deficit: In addition to the occurrence of hallucinations within the use of these systems, requirements traceability is identified as a necessary aspect of safety-critical software engineering, where the life of a requirement is described and followed in both a forward and reverse engineering direction [13]. Furthermore, high-level summaries are often provided by artificial intelligence-supported tools without a mapping to the lines of code or architectural components from which they were derived. As a result, the verification process is complicated by this traceability deficit because artificial intelligence-generated outputs must be manually audited by practitioners to ensure functional alignment with the codebase [19, 57].

3) Semantic loss and contextual reasoning: In addition to these traceability deficits, semantic loss is often suffered by artificial intelligence as the underlying business logic behind a specific implementation is neglected during the translation of code to text [56, 61]. Furthermore, the ability to realize complex nonlinear dependencies across large sources is limited because large language models frequently rely on statistical pattern matching rather than genuine logical deduction [20, 60]. As a result, the behavior of the software system may be captured by extracted requirements, while the critical business constraints or logic that controlled the original design decisions are not documented [8, 61].

B. Comparative Tool Analysis

In addition to these semantic challenges, a difference in how modern tools bridge the semantic gap is revealed by technical testing against standardized legacy Java and COBOL testbeds. Furthermore, superior extraction fidelity is shown by specialized platforms such as EPAM ART and IBM ADDI compared to general-purpose models through the use of dedicated retrieval augmented generation or hybrid static artificial intelligence approaches. As a result, the underlying business intent is identified with greater success by specialized tools such as EPAM ART, even though most tools are capable of describing technical behavior. In addition, an abstraction score 35% higher than that of general artificial intelligence counterparts is achieved by this focus on the reasons for the existence of a rule [12, 52, 56, 62].

Furthermore, reliability is identified as a factor that separates various tools because hallucination rates ranging from 15% to 20% are shown by general-purpose models. Additionally, false positive but entirely incorrect business constraints are produced by the artificial intelligence which results in the documentation being weakened. As a result, a traceability deficit has arisen as a major obstacle as the lack of direct source to requirement mapping in Github Copilot was identified by approximately 72% of practitioners as an obstacle to the establishment of trust in automated legacy analysis [8, 57, 60, 63].

C. Industrial Implications: The SE 3.0 Paradigm

Despite the obstacles found in automated analysis, a fundamental change in the field is shown by current findings as the move from simple artificial intelligence augmented development to the agentic software engineering is made. Furthermore, artificial intelligence supported reverse

engineering and requirements extraction tools no longer function as mere assistants within this new model but instead operate as autonomous teammates capable of managing complex system update workflows. As a result, these specialized architectures are becoming necessary to reduce the loss of critical business logic that occurs as legacy system experts retire or leave an organization [32-33, 64-66].

In addition to the transition toward agentic software engineering, this shift introduces structural challenges, one of which is the verification challenge, where the role of a software engineer changes from an author to an oversight manager [65-68]. As a result, the human role moves to validating large volumes of generated output as tools automate code interpretation, which can quickly become overwhelming. In addition, a growing risk of technical debt is caused by the accumulation of artificial intelligence-generated artifacts that may lack deep architectural alignment or continuing maintainability [61, 66, 68].

Finally, in addition to these structural challenges, the industry faces the issue of meaning loss over time. Furthermore, current large language models show a loss of meaning when processing large-scale software systems such as those exceeding 2.5 million lines of code. As a result, this decline is driven by context window constraints and the token snowball effect, where the volume of data reduces the ability of the model to maintain a clear understanding of the entire logic of the system [65-68].

III. Methodology

A. Research Design

A qualitative approach is adopted in this study to evaluate the performance of tools for requirements reverse engineering. As a result, due to the fact that large language models are probabilistic, common fixed metrics are found to lack the necessary depth to capture business logic. Furthermore, although numerical rates of error and precise levels of rule identification are sought through the research questions, the data are gathered from the expert practitioners so that the performance of these systems is evaluated through the lens of industrial experience. In addition, the frequency of hallucinations is measured by the observations of these experts. Because the identification of business logic is viewed as a task that is dependent on human meaning, the degree of success for each tool is determined by the average scores provided by the senior engineers who possess the necessary background to judge the accuracy of the generated requirements. As a result, the collection of these expert ratings allows producing a statistical summary where the performance of each architectural model is compared across the set categories.

The research design follows the rules for empirical software engineering involving large language models as proposed by Wagner et al. [62]. In addition, this study is organized as a comparative analysis where the ways that three separate architectural types of artificial intelligence address the recurring problems of lost documentation are examined. Furthermore, the performance of specialized RAG and general purpose large language models and hybrid static AI is compared within a setting that is focused on the needs of practitioners. In addition, by focusing on the difficulties of updating old systems, direct value for organizations is provided. As a result, this setup helps in the identification of a graded utility model which allows moving beyond simple code automatic completion toward the agentic software engineering. Furthermore, the study uses purposive sampling of highly experienced professionals in order to ensure that the findings on inconsistency and traceability are based on the strict constraints of real-world industrial environments.

B. Selection of AI-Supported Tools for Requirements Reverse Engineering

Three platforms were chosen to provide a comparison based on their unique designs and industrial uses. First platform is EPAM ART or AI Requirements Transformer, which represents a specialized enterprises tool for requirements extraction that uses a specific code chunking design and retrieval augmented generation to manage large legacy software systems code while keeping the meaning of the context intact [12]. Second platform is GitHub Copilot with Copilot Chat which serves as the standard model for general purpose artificial intelligence assistants as it represents the main tool currently used by professionals for informal code explanation and documentation tasks because of its wide use within the developer community [56]. Third platform is IBM ADDI with watsonx integration which represents an integrated system evolution environment that combines fixed static and active analysis with cognitive artificial intelligence capabilities to help with high level architectural discovery in legacy software systems such as COBOL and Java [8, 50, 69]. Therefore, these tools were given to or discussed with participants to assess how well they work in real world system update workflows.

C. Practitioner Questionnaire Design

Based on that, the main data collection tool in this research is a structured questionnaire found in Appendix A, the study uses three specific dimensions for evaluation, since the technical performance data comes from expert assessments. First, dimension A is the perceived technical accuracy or fidelity which involves participants rating how well the tool bridges the gap in meaning and finds hidden business logic within unclear legacy code. Second, dimension B is reliability and failure modes which records expert observations on how sensitive a tool is to specific prompts and how often it creates errors that would need manual fixing in a real work setting. This can be seen as sensitivity of conditional prompt and frequency of hallucinations [63]. Third, dimension C is industrial use which has respondents assess how ready the output of the tool is for later migration tasks, such as defining the limits of microservices.

D. Data Collection and Analysis

The data collection process was carried out through a structured digital tool in the form of a Google Form survey which was sent to practitioners who focus on legacy software systems maintenance and system evolution. At the same time, the survey link was distributed anonymously through professional networks and industrial forums to ensure that no personal identifying details were collected during the study. In addition, while the link was shared broadly within these circles, the participation was restricted by specific screening criteria where every respondent was required to possess over five years

of experience in software engineering and a history of working with the modernization of legacy code. Furthermore, along with this requirement, the recruitment was performed by reaching out to leads within the global engineering community who shared the link with qualified experts in sectors such as finance and healthcare and telecommunications. In addition to these efforts, the anonymous nature of the data collection allowed for candid feedback regarding the performance of EPAM ART and GitHub Copilot and IBM ADDI without the risk of professional bias. Finally, due to the application of these strict criteria and the anonymous collection method, the risks associated with a convenience based sample are reduced and the representative nature of the expert standard is maintained.

IV. Results and Discussion

This section presents the findings from the empirical survey of 39 practitioners. Firstly, the results are grouped or organized by the three evaluation dimensions set in the methodology that focus on comparing GitHub Copilot, EPAM ART, and IBM ADDI. Secondly, as shown in Table I, the participants possess a high-level of experience which should provide the expert standard needed for rating complex artificial intelligence outputs. Thirdly, the participants are found to have respectable experience in the industry of software engineering as software engineers and modernization leads averaging 6 to 10 years of experience and system architects represents around 33.3% of the sample and average 11 to 15 years in the field. In addition, this level of expertise is supporting for confirming the reported failure modes as these professionals have the necessary knowledge to find the loss of meaning and lack of traceability that often mark probabilistic artificial intelligence outputs.

TABLE I. Participants Demographic Profile (N=39)

Professional Role	Participants (n)	Percentage (%)	Avg. Experience (Years)
Senior Software Engineers	19	48.7%	6-10
System Architects	13	33.3%	11-15
Modernization Leads	7	18.0%	6-10
Total	39	100%	> 8 years

A. Dimension A: Technical Fidelity and Abstraction

Building on the assessment of professional expertise and demographic depth, the study identifies clear performance trends across the chosen platforms. Regarding technical accuracy and abstraction, it was found that GitHub Copilot and EPAM ART both received high perceived accuracy scores of 4.26 out of 5 from the participants. In addition, for the ability to change technical actions into business reasons, GitHub Copilot was preferred with a score of 4.05. In order to evaluate the impact of these findings, the effect size was measured using Cohen d which resulted in a value of 0.78 for the comparison between general purpose and specialized tools. Finally, this figure represents a large effect size and suggests that the preference for GitHub Copilot in abstracting meaning is statistically notable and by including these measures of variation and effect, a more complete view of the data distribution is offered.

B. Dimension B: Reliability and Failure Modes

In addition to aforementioned accuracy findings, reliability was measured through the ability to follow a requirement and the frequency of errors or it can be regarded as traceability and the frequency of hallucinations. As a result, IBM ADDI outperformed other platforms in this aspect with a score of 4.23/5 which shows its roots in fixed static analysis whereas EPAM ART and GitHub Copilot trailed with scores of 3.92/5 and 3.82/5. Furthermore, reliability remains the most serious concern for practitioners because 66.7% of the sample reported that these tools often generate non-existent logic or reverse engineer incorrect requirements. Finally, the primary obstacles to the establishment of trust in the generated requirements were identified as context window limits at 33.3% and factual errors at 28.2% along with a lack of formal traceability at 23.1%. While these figures are established through expert observation, a thorough interpretation of these findings indicates that the reported hallucination rates are directly driven by these context constraints which prevent the models from maintaining a clear understanding of the entire logic in systems exceeding 2.5 million lines of code. Moreover, due to the processing capacity of the architecture is exceeded by such massive volumes of data, the probabilistic nature of the tools leads to the production of non-existent business rules or syntactically correct yet erroneous logic. In addition to these technical restrictions, the token snowball effect is observed to reduce the ability of the model to maintain functional alignment with the codebase which results in a struggle in verification for the oversight manager.

C. Dimension C: Industrial Utility

Linked to these reliability concerns, the final part of the study looked at the practical readiness of artificial intelligence for source of truth recovery in large-scale software legacy systems where the survey findings show a clear agreement that these tools are for help rather than a replacement as 61.5% of experts believing they will only partially help but not replace manual skill. However, a notable 38.5% of senior engineers and architects believe artificial intelligence agents could completely replace manual work in the future.

D. Discussion of Findings

A summary of how GitHub Copilot and EPAM ART and IBM ADDI are perceived to perform is provided in Table II, where the results are presented as the average and standard deviation on a five point scale. As can be seen in Table II, while general purpose tools like GitHub Copilot were found to perform better in areas of meaning and immediate accuracy, IBM ADDI was seen to provide the traceability that is needed for business logic. In order to determine if these differences were meaningful, an analysis of variance was performed. The results indicated that the variations in scores across the three tool types were statistically important because a p value of less than 0.05 was achieved. Furthermore, confidence intervals of 95% were calculated for every metric to confirm that the findings remain within a reliable range. Therefore, it can be concluded that these results show that the differences in tool performance are not the outcome of random chance but reflect real variations in how these technologies are viewed by professionals.

Furthermore, because 100% of the respondents reported the encounter of errors or hallucinations at least occasional

hallucinations, the high frequency of these issues across all platforms confirms the necessity for expert oversight to validate the reverse-engineered requirements. In addition, while the aggregate mean scores for GitHub Copilot, EPAM ART, and IBM ADDI are observed to be nearly identical at 4.04 and 4.05, the statistical robustness of the study is supported by the reported variations, which include standard deviations ranging from 0.72 to 0.91. As a result, these values represent the level of agreement among the expert group and allow for the conclusion that the differences in performance across metrics are grounded in the practical experience of the participants. In addition to these metrics, the high traceability score of 4.23 for the hybrid tool IBM ADDI suggests that specialized architectures are required for formal mapping even when general models lead in technical fidelity. Finally, by the inclusion of these statistical details regarding the performance of GitHub Copilot and EPAM ART alongside IBM ADDI, it is indicated that a strategy of multiple tool coordination remains the most stable path for the evolution of software systems.

TABLE II. COMPARATIVE PERFORMANCE SCORES (N=39) SCALE 1–5

Metric	GitHub Copilot	EPAM ART	IBM ADDI
Extraction Fidelity	4.26±0.88	4.26±0.91	3.92±0.72
Semantic Abstraction	4.05±0.86	3.97±0.81	4.00±0.86
Traceability	3.82±0.79	3.92±0.77	4.23±0.81
Aggregate Mean Score	4.04 ±0.85	4.05 ±0.84	4.05±0.81

V. CONCLUSION AND FUTURE WORK

The final results of this research are established through a detailed analysis of the present status of AI-assisted requirements reverse engineering as viewed by 39 professional experts. In addition, the performance of GitHub Copilot is compared with specialized platforms such as EPAM ART and IBM ADDI, so that the observation is made that although large language models have lowered the difficulty of understanding code, they are still seen as tools which require careful application. Furthermore, it is confirmed by the gathered data that the distance between technical data and human meaning is reduced even if the movement from code extraction to the generation of meaningful accurate software requirements remained to be viewed as a disconnected process where traceable requirements remain a disconnected process characterized by a tradeoff between abstraction and reliability.

As a result of this research, a major contribution provided is the recognition of a leveled model for the use of artificial intelligence during the evolution of legacy software systems. Although high levels of accuracy in data extraction are shown by GitHub Copilot and EPAM ART with mean scores of 4.26 plus or minus 0.88 and 4.26 plus or minus 0.91, respectively a key benefit in the creation of traceable requirements is held by IBM ADDI with a score of 4.23 plus or minus 0.81. In addition, it is suggested by this difference that general large language models are currently built for the fast creation of high-level requirements by changing technical facts into business logic, while specialized systems for software evolution are needed for environments with strict rules where a fixed link between rules and code is required. Moreover, the fact that false data or hallucinations are reported as a regular or periodic event by experts serves as evidence that the use of artificial intelligence for requirements engineering is viewed as a supporting tool rather than a total change or human replacement.

Furthermore, a shared view is found among professionals, with 79.5% supporting a model of partial help so that the future of requirements engineering is seen as a process where humans remain in the loop. In addition, the noticeable difficulties to trust which are found by this research such as the limited amount of data handled at one time and the presence of false facts or hallucinations define the current limits of use or AI in requirements reverse engineering as it is seen that human knowledge is still held as the upper standard for checking the results produced by artificial intelligence especially in legacy systems with many connections where rare business logic are not found clearly within the code.

Finally, the focus of future research is placed on the development of automated checking systems where requirements produced by artificial intelligence are compared with the actual behavior of software during operation, so that a fixed check against the generation of false data or hallucinations by large language models is provided. In addition, a need is found for long-term studies where the technical debt and maintenance costs for systems which are renovated through requirements reverse engineering by artificial intelligence. Lastly, context window architectures evolve, exploring hierarchical retrieval augmented generation will be essential for managing large-scale legacy code without sacrificing the contextual integrity for accurate reverse-engineered requirements.

ACKNOWLEDGMENT

I would like to express my sincere gratitude to Umm Al Qura University for providing me with the invaluable opportunity to conduct research in the software engineering field. I would also like to extend my heartfelt thanks to my family and friends for their unwavering support and encouragement throughout this journey.

REFERENCES

[1] I. Sommerville, Software Engineering, 10th edition. Boston: Pearson, 2015.

[2] E. Aghajani *et al.*, "Software documentation issues unveiled," in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, IEEE, 2019, p. 1199. doi: 10.1109/ICSE.2019.00122.

[3] E. Aghajani *et al.*, "Software documentation: the practitioners' perspective," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, Seoul South Korea: ACM, Jun. 2020, pp. 590–601. doi: 10.1145/3377811.3380405.

[4] A. A. H. Alzahrani, "Software Systems Documentation: A Systematic Review," *Int. J. Adv. Comput. Sci. Appl.*, vol. 15, no. 8, pp. 155–162, 2024, doi: 10.14569/IJACSA.2024.0150816.

[5] R. Kalantari and T. C. Lethbridge, "Unveiling Developers' Mindset Barriers to Software Modeling Adoption," in *2023 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*, Västerås, Sweden: IEEE, Oct. 2023, pp. 737–746. doi: 10.1109/MODELS-C59198.2023.00120.

[6] A. S. M. Venigalla and S. Chimalakonda, "Understanding Emotions of Developer Community Towards Software Documentation," in *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Society (ICSE-SEIS)*, Madrid, ES: IEEE, May 2021, pp. 87–91. doi: 10.1109/ICSE-SEIS52602.2021.00018.

[7] E. J. Chikofsky and J. H. Cross, "Reverse engineering and design recovery: A taxonomy," *IEEE Softw.*, vol. 7, no. 1, pp. 13–17, 2002.

[8] A. Ghimire, J. Zhang, S. R. Lingala, F. Alsulami, and F. Amsaad, "A Survey on Application of AI on Reverse Engineering for Software Analysis and Security," *IEEE Access*, pp. 152903–152913, 2025.

[9] M. Alenezi and M. Akour, "AI-Driven Innovations in Software Engineering: A Review of Current Practices and Future Directions," *Appl. Sci.*, vol. 15, no. 3, p. 1344, Jan. 2025, doi: 10.3390/app15031344.

[10] J. M. Carrillo-de-Gea, C. Ebert, M. Hosni, A. Vizcaíno, J. Nicolás, and J. L. Fernández-Alemán, "Tools for Requirements Engineering," *IEEE Softw.*, vol. 41, no. 04, pp. 30–37, Jul. 2024, doi: 10.1109/MS.2024.3385466.

[11] C. Ebert and P. Louridas, "Generative AI for Software Practitioners," *IEEE Softw.*, vol. 40, no. 4, pp. 30–38, Jul. 2023, doi: 10.1109/MS.2023.3265877.

[12] X. Hou *et al.*, "Large Language Models for Software Engineering: A Systematic Literature Review," *ACM Trans. Softw. Eng. Methodol.*, vol. 33, no. 8, pp. 1–79, Nov. 2024, doi: 10.1145/3695988.

[13] A. Hemmat, M. Sharbaf, S. Kolahdouz-Rahimi, K. Lano, and S. Y. Tehrani, "Research directions for using LLM in software requirement engineering: A systematic review," *Front. Comput. Sci.*, vol. 7, p. 1519437, 2025.

[14] G. P. Reddy, Y. V. Pavan Kumar, and K. P. Prakash, "Hallucinations in Large Language Models (LLMs)," in *2024 IEEE Open Conference of Electrical, Electronic and Information Sciences (eStream)*, Vilnius, Lithuania: IEEE, Apr. 2024, pp. 1–6. doi: 10.1109/eStream61684.2024.10542617.

[15] R. Massenon, I. Gambo, J. A. Khan, C. Agbonkhese, and A. Alwadain, "'My AI is Lying to Me': User-reported LLM hallucinations in AI mobile apps reviews.," *Sci. Rep.*, vol. 15, no. 1, pp. 1–15, Aug. 2025, doi: 10.1038/s41598-025-15416-8.

[16] Z. Zhang *et al.*, "LLM Hallucinations in Practical Code Generation: Phenomena, Mechanism, and Mitigation," *Proc. ACM Softw. Eng.*, vol. 2, no. ISSTA, pp. 481–503, Jun. 2025, doi: 10.1145/3728894.

[17] Y. Bang *et al.*, "HalluLens: LLM Hallucination Benchmark," in *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Vienna, Austria: Association for Computational Linguistics, 2025, pp. 24128–24156. doi: 10.18653/v1/2025.acl-long.1176.

[18] L. Huang *et al.*, "A Survey on Hallucination in Large Language Models: Principles, Taxonomy, Challenges, and Open Questions," *ACM Trans. Inf. Syst.*, vol. 43, no. 2, pp. 1–55, Mar. 2025, doi: 10.1145/3703155.

[19] H. Legesse and G. Bicknell, "AI-Enhanced Requirements Traceability Using MBSE and LLMs for Complex Systems," in *AI4SE & SE4AI Research and Application Workshop*, 2025. [Online]. Available: https://sercuarc.org/event/ai4se-se4ai-workshop-2025/

[20] J. S. Bhalla and M. K. Jodhka, "Leveraging Large Language Models in the Software Development Lifecycle: Opportunities and Challenges.," *Int. J. Adv. Comput. Sci. Appl.*, vol. 16, no. 11, p. 43, 2025.

[21] H. van Vliet, *Software Engineering: Principles and Practice*. Chichester: John Wiley & Sons Inc, 2008.

[22] H. M. Sneed, "A cost model for software maintenance & evolution," in *20th IEEE International Conference on Software Maintenance, 2004. Proceedings.*, Chicago, IL, USA: IEEE, 2004, pp. 264–274. doi: 10.1109/ICSM.2004.1357810.

[23] H. C. Benestad, B. Anda, and E. Arisholm, "Understanding software maintenance and evolution by analyzing individual changes: a literature review," *J. Softw. Maint. Evol. Res. Pract.*, vol. 21, no. 6, pp. 349–378, Nov. 2009, doi: 10.1002/smr.412.

[24] N. Chapin, J. E. Hale, K. Md. Khan, J. F. Ramil, and W. Tan, "Types of software evolution and software maintenance," *J. Softw. Maint. Evol. Res. Pract.*, vol. 13, no. 1, pp. 3–30, Jan. 2001, doi: 10.1002/smr.220.

[25] V. Rajlich, "Software evolution and maintenance," in *Future of Software Engineering Proceedings*, Hyderabad India: ACM, May 2014, pp. 133–144. doi: 10.1145/2593882.2593893.

[26] P. Tripathy and K. Naik, *Software Evolution and Maintenance: A Practitioner's Approach*, 1st ed. Wiley, 2014. doi: 10.1002/9781118964637.

[27] H. A. Siala, "Enhancing Model-Driven Reverse Engineering Using Machine Learning," in *Proceedings of the 2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings*, Lisbon Portugal: ACM, Apr. 2024, pp. 173–175. doi: 10.1145/3639478.3639797.

[28] I. P. Uppala, "Microservices and Cloud-Native Platforms: Transforming the Insurance Industry," *Eur. J. Comput. Sci. Inf. Technol.*, vol. 13, no. 50, pp. 34–44, Jul. 2025.

[29] K. Henderson and A. Salado, "Value and benefits of model-based systems engineering (MBSE): Evidence from the literature," *Syst. Eng.*, vol. 24, no. 1, pp. 51–66, Jan. 2021, doi: 10.1002/sys.21566.

[30] K. Manikas and K. M. Hansen, "Software ecosystems – A systematic literature review," *J. Syst. Softw.*, vol. 86, no. 5, pp. 1294–1306, May 2013, doi: 10.1016/j.jss.2012.12.026.

[31] M. F. Lungu, "Reverse engineering software ecosystems," Università della Svizzera italiana, 2009. [Online]. Available: https://susi.usi.ch/usi/documents/318234

[32] A. E. Hassan, G. A. Oliva, D. Lin, B. Chen, and Z. M. (Jack) Jiang, "Towards AI-Native Software Engineering (SE 3.0): A Vision and a Challenge Roadmap," *ACM Trans. Softw. Eng. Methodol.*, p. 3807901, Apr. 2026, doi: 10.1145/3807901.

[33] F. R. Cogo, G. A. Oliva, and A. E. Hassan, "Compiler.next: A Search-Based Compiler to Power the AI-Native Future of Software Engineering," *ACM Trans. Softw. Eng. Methodol.*, p. 3802581, Mar. 2026, doi: 10.1145/3802581.

[34] R. Mall and PHI Learning, *Fundamentals of software engineering*, Fifth edition. in Eastern Economy Edition. Delhi: PHI Learning Private Limited, 2018.

[35] L. Zhao *et al.*, "Natural Language Processing for Requirements Engineering: A Systematic Mapping Study," *ACM Comput. Surv.*, vol. 54, no. 3, pp. 1–41, Apr. 2022, doi: 10.1145/3444689.

[36] Y. Yang, X. Xia, D. Lo, and J. Grundy, "A Survey on Deep Learning for Software Engineering," *ACM Comput. Surv.*, vol. 54, no. 10s, pp. 1–73, Jan. 2022, doi: 10.1145/3505243.

[37] P. A. Laplante and M. H. Kassab, *Requirements Engineering for Software and Systems*, 4th ed. New York: Auerbach Publications, 2022. doi: 10.1201/9781003129509.

[38] D. Gobov and O. Zuieva, "Software Quality Attributes in Requirements Engineering," *Int. J. Inf. Technol. Comput. Sci.*, vol. 17, no. 4, pp. 38–48, Aug. 2025, doi: 10.5815/ijitcs.2025.04.04.

[39] T. Hovorushchenko, D. Medzatyi, Y. Voichur, and M. Lebiga, "Method for forecasting the level of software quality based on quality attributes," *J. Intell. Fuzzy Syst.*, vol. 44, no. 3, pp. 3891–3905, Mar. 2023, doi: 10.3233/JIFS-222394.

[40] B. I. Belinda, A. Akintoba, N. Solomon, and A. Boniface, "Evaluating Software Quality Attributes using Analytic Hierarchy Process (AHP)," *Int. J. Adv. Comput. Sci. Appl.*, vol. 12, no. 3, 2021, doi: 10.14569/IJACSA.2021.0120321.

[41] G. Carrozza, R. Pietrantuono, and S. Russo, "A software quality framework for large-scale mission-critical systems engineering," *Inf. Softw. Technol.*, vol. 102, pp. 100–116, Oct. 2018, doi: 10.1016/j.infsof.2018.05.009.

[42] R. Li, P. Liang, M. Soliman, and P. Avgeriou, "Understanding software architecture erosion: A systematic mapping study," *J. Softw. Evol. Process*, vol. 34, no. 3, p. e2423, Mar. 2022, doi: 10.1002/smr.2423.

[43] A. Baabad, H. B. Zulzalil, S. Hassan, and S. B. Baharom, "Characterizing the Architectural Erosion Metrics: A Systematic Mapping Study," *IEEE Access*, vol. 10, pp. 22915–22940, 2022, doi: 10.1109/ACCESS.2022.3150847.

[44] R. Li, P. Liang, M. Soliman, and P. Avgeriou, "Understanding Architecture Erosion: The Practitioners' Perceptive," in *2021 IEEE/ACM 29th International Conference on Program Comprehension (ICPC)*, Madrid, Spain: IEEE, May 2021, pp. 311–322. doi: 10.1109/ICPC52881.2021.00037.

[45] A. Baabad, H. B. Zulzalil, S. Hassan, and S. B. Baharom, "Software Architecture Degradation in Open Source Software: A Systematic Literature Review," *IEEE Access*, vol. 8, pp. 173681–173709, 2020, doi: 10.1109/ACCESS.2020.3024671.

[46] S.-Y. Yu, Y. G. Achamyeleh, C. Wang, A. Kocheturov, P. Eisen, and M. A. Al Faruque, "CFG2VEC: Hierarchical Graph Neural Network for Cross-Architectural Software Reverse Engineering," in *2023 IEEE/ACM 45th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, Melbourne, Australia: IEEE, May 2023, pp. 281–291. doi: 10.1109/ICSE-SEIP58684.2023.00031.

[47] T. Faingnaert et al., "Tools and Models for Software Reverse Engineering Research," in Proceedings of the 2024 Workshop on Research on offensive and defensive techniques in the context of Man At The End (MATE) attacks, Salt Lake City UT USA: ACM, Nov. 2024, pp. 44–58. doi: 10.1145/3689934.3690817.

[48] N. Anquetil *et al.*, "Modular Moose: A New Generation of Software Reverse Engineering Platform," in *Reuse in Emerging Software Engineering Practices*, vol. 12541, S. Ben Sassi, S. Ducasse, and H. Mili, Eds., in Lecture Notes in Computer Science, vol. 12541. , Cham: Springer International Publishing, 2020, pp. 119–134. doi: 10.1007/978-3-030-64694-3_8.

[49] F. Buonamici, M. Carfagni, R. Furferi, L. Governi, A. Lapini, and Y. Volpe, "Reverse engineering modeling methods and tools: a survey," *Comput.-Aided Des. Appl.*, vol. 15, no. 3, pp. 443–464, May 2018, doi: 10.1080/16864360.2017.1397894.

[50] E. Putrycz and A. W. Kark, "Recovering Business Rules from Legacy Source Code for System Modernization," in *Advances in Rule Interchange and Applications*, vol. 4824, A. Paschke and Y. Biletskiy, Eds., in Lecture Notes in Computer Science, vol. 4824. , Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 107–118. doi: 10.1007/978-3-540-75975-1_9.

[51] H. Washizaki, *Guide to the Software Engineering Body of Knowledge v4.0a*. IEEE Computer Society, 2024. Accessed: Jan. 03, 2026. [Online]. Available: https://ieeecs-media.computer.org/media/education/swebok/swebok-v4.pdf

[52] J. Ullrich, M. Koch, and A. Vogelsang, "From requirements to code: Understanding developer practices in llm-assisted software engineering," in *2025 IEEE 33rd International Requirements Engineering Conference (RE)*, IEEE, 2025, pp. 257–266. doi: 10.1109/RE63999.2025.00032.

[53] A. Patterson, "Large language models (LLMs) in systems engineering and design," Jul. 08, 2024. doi: 10.36227/techrxiv.172047441.13201097/v1.

[54] I. Ozkaya, "Application of Large Language Models to Software Engineering Tasks: Opportunities, Risks, and Implications," *IEEE Softw.*, vol. 40, no. 3, pp. 4–8, May 2023, doi: 10.1109/MS.2023.3248401.

[55] A. Fan *et al.*, "Large Language Models for Software Engineering: Survey and Open Problems," in *2023 IEEE/ACM International Conference on Software Engineering: Future of Software Engineering (ICSE-FoSE)*, Melbourne, Australia: IEEE, May 2023, pp. 31–53. doi: 10.1109/ICSE-FoSE59343.2023.00008.

[56] V. Terragni, A. Vella, P. Roop, and K. Blincoe, "The Future of AI-Driven Software Engineering," *ACM Trans. Softw. Eng. Methodol.*, vol. 34, no. 5, pp. 1–20, Jun. 2025, doi: 10.1145/3715003.

[57] C. Ge, T. Wang, X. Yang, and C. Treude, "Cross-Level Requirements Tracing Based on Large Language Models," *IEEE Trans. Softw. Eng.*, 2025, doi: 10.1109/TSE.2025.3572094.

[58] H. Jin, L. Huang, H. Cai, J. Yan, B. Li, and H. Chen, "From LLMs to LLM-based Agents for Software Engineering: A Survey of Current, Challenges and Future," Apr. 13, 2025, *arXiv*: arXiv:2408.02479. doi: 10.48550/arXiv.2408.02479.

[59] J. A. H. López, B. Chen, M. Saad, T. Sharma, and D. Varró, "On inter-dataset code duplication and data leakage in large language models," *IEEE Trans. Softw. Eng.*, 2024, doi: 10.1109/TSE.2024.3504286.

[60] R. Gröpler *et al.*, "The Future of Generative AI in Software Engineering: A Vision from Industry and Academia in the European GENIUS Project," in *IEEE*, arXiv, Nov. 2025. doi: 10.48550/arXiv.2511.01348.

[61] M. A. Abbasi, P. Ihantola, T. Mikkonen, and N. Mäkitalo, "Reconsidering Requirements Engineering: Human–AI Collaboration in AI-Native Software Development," in *Software Engineering and Advanced Applications*, D. Taibi and D. Smite, Eds., Cham: Springer Nature Switzerland, 2025, pp. 164–180. doi: 10.1007/978-3-032-04190-6_11.

[62] S. Wagner, M. M. Barón, D. Falessi, and S. Baltes, "Towards Evaluation Guidelines for Empirical Studies Involving LLMs," in *2025 IEEE/ACM International Workshop on Methodological Issues with Empirical Studies in Software Engineering (WSESE)*, May 2025, pp. 24–27. doi: 10.1109/WSESE66602.2025.00011.

[63] H. A. Dang, V. Tran, and L.-M. Nguyen, "Survey and analysis of hallucinations in large language models: attribution to prompting strategies or model behavior," *Front. Artif. Intell.*, vol. 8, p. 1622292, 2025.

[64] A. E. Hassan *et al.*, "Agentic Software Engineering: Foundational Pillars and a Research Roadmap," Sep. 23, 2025, *arXiv*: arXiv:2509.06216. doi: 10.48550/arXiv.2509.06216.

[65] A. K. Gangula, "A COMPARATIVE ANALYSIS OF LLM-DRIVEN VS. MANUAL LEGACY CODE REFACTORING: A CASE STUDY IN .NET CORE MIGRATION," *Int. J. Comput. Sci. Mob. Comput.*, vol. 14, no. 9, pp. 80–91, Sep. 2025, doi: 10.47760/ijcsmc.2025.v14i09.012.

[66] A. Roychoudhury, "Agentic AI for Software: thoughts from Software Engineering community," Sep. 22, 2025, *arXiv*: arXiv:2508.17343. doi: 10.48550/arXiv.2508.17343.

[67] M. Binz *et al.*, "How should the advancement of large language models affect the practice of science?," *Proc. Natl. Acad. Sci.*, vol. 122, no. 5, p. e2401227121, Feb. 2025, doi: 10.1073/pnas.2401227121.

[68] L. Solovyeva, E. Carneiro Oliveira, S. Fan, A. Tuncay, S. Gareev, and A. Capiluppi, "Leveraging LLMs for Automated Translation of Legacy Code: A Case Study on PL/SQL to Java Transformation," in *Proceedings of the 29th International Conference on Evaluation and Assessment in Software Engineering*, Istanbul Turkiye: ACM, Jun. 2025, pp. 809–813. doi: 10.1145/3756681.3757007.

[69] B. Krishnaswamy Gnanasekaran, "AI-AUGMENTED LEGACY SYSTEM INTERPRETATION: RESEARCH SUMMARY AND NEW DIRECTION," *Int. J. Artif. Intell. Res. Dev.*, vol. 3, no. 2, pp. 109–125, Sep. 2025, doi: 10.34218/IJAIRD_03_02_007.

APPENDIX A

Survey: AI-Supported Requirements Reverse Engineering (RRE)

Research Study: Software Reverse Engineering Using AI Platforms. This survey evaluates the efficacy of AI tools in extracting business logic from legacy systems.

* Indicates required question

1. Current Professional Role: *

Mark only one oval.

- ◯ Senior Software Engineer
- ◯ System Architect
- ◯ Modernization/Digital Transformation Lead
- ◯ Other

2. Years of Experience in Software Maintenance/Modernization: *

Mark only one oval.

- ◯ 0–5 years
- ◯ 6–10 years
- ◯ 11–15 years
- ◯ 15+ years

3. Familiarity with Legacy Environments (COBOL, Java Monoliths, etc.): *

Mark only one oval.

- ◯ Expert
- ◯ Proficient
- ◯ Competent
- ◯ Novice

4. Which tools have you utilized for code comprehension or documentation? (Check all that apply): *

Check all that apply.

- ☐ GitHub Copilot
- ☐ EPAM ART
- ☐ IBM ADDI (watsonx)
- ☐ Other LLMs (GPT-4, Claude, Llama)

5. Extraction Fidelity: How accurately does the tool extract core business rules from code? (1 = Low Accuracy, 5 = High Accuracy) *

Mark only one oval per row.

	1	2	3	4	5
GitHub Copilot	◯	◯	◯	◯	◯
EPAM ART	◯	◯	◯	◯	◯
IBM ADDI	◯	◯	◯	◯	◯

6. Semantic Abstraction: Ability to translate technical 'What' into business 'Why.' (1 = Purely Technical, 5 = High-Level Business Logic) *

Mark only one oval per row.

	1	2	3	4	5
GitHub Copilot	◯	◯	◯	◯	◯
EPAM ART	◯	◯	◯	◯	◯
IBM ADDI	◯	◯	◯	◯	◯

7. Traceability: Does the tool provide clear mapping from requirement to source line? (1 = Black Box, 5 = Full Source Mapping) *

Mark only one oval per row.

	1	2	3	4	5
GitHub Copilot	◯	◯	◯	◯	◯
EPAM ART	◯	◯	◯	◯	◯
IBM ADDI	◯	◯	◯	◯	◯

8. Hallucination Frequency: How often does the tool generate non-existent logic? *

Mark only one oval.

- ◯ Never (<1%)
- ◯ Rare (1–10%)
- ◯ Occasional (10–20%)
- ◯ Frequent (>20%)

9. In your view, what is the primary barrier to trusting AI-generated requirements? *

Mark only one oval.

- ◯ Factual Inconsistency/Hallucinations
- ◯ Lack of formal traceability
- ◯ Context window limitations in large codebases
- ◯ Security/Privacy concerns

10. Describe a scenario where AI-driven RRE failed to capture critical business logic.

11. Do you believe AI agents can eventually replace manual 'source of truth' recovery? *

Mark only one oval.

- ◯ Yes, completely
- ◯ Partially, they will assist but not replace
- ◯ No, manual expertise will remain essential