



WHEN CLEAN CODE BECOMES COSTLY: BALANCING HUMAN READABILITY AND AI DIGESTIBILITY WITH THE NUDEE FRAMEWORK

Wachiranun Pum	Faculty of Management Science and Information Technology, Nakhon Phanom University, Nakhon Phanom, Thailand	wachiranun@npu.ac.th
Narongsak Sukma*	Faculty of Digital, Siam Technology College, Bangkok, Thailand	narongsaks@siamtechno.ac.th

* Corresponding author

ABSTRACT

Aim/Purpose	This study maps the structural tension between human-centric Clean Code conventions and the architectural demands of LLM-based coding agents. It proposes NUDEE, a management framework for governing codebases optimized for both human readability and AI digestibility.
Background	Clean Code prescribes deep abstraction and module decomposition to reduce human memory load. By 2026, LLMs and autonomous agents actively participate in coding workflows, but their token-limited context windows are penalized by the same decomposition that helps human developers.
Methodology	A PRISMA 2020-guided systematic literature review of three databases (Scopus, Web of Science, ACM Digital Library) used a three-pillar Boolean search strategy followed by four-stage screening. The resulting corpus of 89 records (1996–2026) was inductively coded into five thematic streams.
Contribution	The paper presents a PRISMA-structured systematic literature review – to our knowledge, one of the first to systematically map the human-readability–AI digestibility tension in software engineering – documents the field’s organizational turn (44.9% of the literature addresses management frameworks) and introduces the NUDEE framework.

Accepting Editor Ahmad Samed Al-Adwan | Received: June 1, 2026 | Revised: July 25, July 27,
August 5, 2026 | Accepted: August 7, 2026.

Cite as: Pum, W., & Sukma, N. (2026). When clean code becomes costly: Balancing human readability and AI digestibility with the NUDEE framework. *Interdisciplinary Journal of Information, Knowledge, and Management*, 21, Article 18. <https://doi.org/10.28945/5866>

(CC BY-NC 4.0) This article is licensed to you under a [Creative Commons Attribution-NonCommercial 4.0 International License](https://creativecommons.org/licenses/by-nc/4.0/). When you copy and redistribute this paper in full or in part, you need to provide proper attribution to it to ensure that others can later locate this work (and to ensure that others do not accuse you of plagiarism). You may (and we encourage you to) adapt, remix, transform, and build upon the material for any non-commercial purposes. This license does not permit you to use this material for commercial purposes.

Findings	The dominant stream, hybrid and agentic SE management frameworks (44.9%), confirms that the field has reframed the problem as organizational; synthesis across all streams reveals three structural pillars for AI-optimal codebases.
Recommendations for Practitioners	Development teams should adopt the Software Coach and System Orchestrator roles, instrument the Context Efficiency Index in CI/CD pipelines to detect semantic debt early, and revise code review criteria to include digestibility alongside correctness.
Recommendations for Researchers	Empirical validation of the CEI against real agent task-solve rates in production codebases is the field’s most pressing research need.
Impact on Society	As AI agents increasingly execute code in critical systems, semantic debt in organizational codebases directly translates into reliability and safety risk. NUDEE provides a governance vocabulary that enables mid-size organizations, not only large technology firms, to manage human AI software knowledge assets responsibly.
Future Research	CEI empirical validation with threshold calibration and longitudinal case studies testing whether teams adopting the dual-role model achieve higher agent reliability and shorter release cycles.
Keywords	software architecture, knowledge management, large language models, context efficiency, sociotechnical systems, AI digestibility

INTRODUCTION

Contemporary software engineering is characterized by a structural paradox. The architectural prescriptions that make codebases most readable for human developers (decompose functions aggressively, keep modules small, distribute concerns across many files) are precisely those that impose the highest retrieval and attention costs on large language models. Decomposition is good for human working memory. Decomposition is bad for LLM context windows. The design choice that reduces a developer’s cognitive load fragments the semantic context an agent needs to reason about a modification task. This paradox is not a short-term artifact of immature tooling; it is an architectural incompatibility between two cognitively distinct classes of agent, and it demands a management response.

Its practical urgency has surfaced rapidly. Human development teams were trained to follow Clean Code (Martin, 2008), SOLID principles, and agile decomposition. By 2026, LLMs and autonomous coding agents will be involved in code generation, review, and refactoring at an organizational scale (Fan et al., 2023; He et al., 2025; Otoum & Elkhilili, 2026; Siddeeq et al., 2026). These agents work best on codebases whose files are locally self-contained, semantically dense, and structurally consolidated. This profile is at odds with almost all of the Clean Code prescriptions at almost every point. The empirical evidence for this conflict is accumulating rapidly. Yet, the literature on this subject has not been systematically synthesized, and no management framework has been developed to translate the technical findings into organizational governance. This paper fills the gap. The research is guided by three research questions.

RQ1: How do traditional human-centric software design patterns influence the retrieval and generation performance of LLMs for automated software maintenance?

RQ2: What architectural and methodological changes are emerging in the literature to make codebases more digestible for AI coding agents?

RQ3: How can organizations systematically measure and manage the balance between human readability and machine digestibility to maximize the value of their software knowledge asset?

To address these questions, this paper makes three contributions. First, it reports one of the first structured systematic literature reviews of the PRISMA 2020 literature on the human readability–AI digestibility tension in software engineering (the extent to which a codebase’s structure allows an autonomous coding agent to retrieve and act on it accurately within a limited context window) synthesizing 89 peer-reviewed records (1996–2026) from Scopus, Web of Science and ACM Digital Library. Second, it documents the field’s organizational turn: 44.9% of the corpus addresses management and governance frameworks rather than technical solutions, signaling that practitioners have already recognized this as an organizational design problem. Third, it proposes NUDEE (Navigating hUman–AI Digestibility for Efficient Engineering), a management framework grounded in double-loop learning (Argyris & Schön, 1978) and sociotechnical systems theory (Emery & Trist, 1960) that translates these empirical findings into practice: two developer roles that reorganize who is responsible for a codebase’s teachability, three structural engineering pillars that specify how code should be organized, and a measurable index (Context Efficiency Index, CEI) for governing human–AI codebase quality.

The remainder of the paper is organized as follows. The next section develops the theoretical background, synthesizing three lenses into the scaffolding for NUDEE and informing RQ1, and then describes the PRISMA 2020 methodology. The five-stream synthesis of the literature is then presented, which, together with the previous section, answers RQ1 and RQ2. The NUDEE framework is presented, addressing RQ3. The theoretical and practical implications are then discussed, and threats to validity are addressed. Finally, a specific research agenda is presented.

THEORETICAL BACKGROUND

The NUDEE framework integrates three theoretical lenses that together explain why the human–AI readability tension is not merely a technical curiosity but an organizational design problem. Cognitive load theory explains why Clean Code’s prescriptions are optimal for human developers; information processing theory explains why those same prescriptions impose retrieval penalties on LLMs; and knowledge management theory explains why the resulting tension carries strategic consequences for organizations that embed AI agents in their development workflows.

COGNITIVE LOAD THEORY AND THE ARCHITECTURE OF ABSTRACTION

According to cognitive load theory (Sweller, 1988), the limited capacity of working memory constrains human problem solving. In software engineering, the primary cognitive bottleneck is code comprehension: a developer reading an unfamiliar function must simultaneously hold its syntax, purpose, and dependency relationships in active memory. The Clean Code prescriptions, short functions, single-responsibility modules, and descriptive naming, are cognitively optimal responses to this bottleneck. They reduce the number of items held in working memory at any given time, enabling developers to reason about complex systems by controlling the complexity of each local unit.

The limiting factor for LLMs is not working memory. The effective ‘working memory’ of an LLM is its context window, the token sequence available in a single forward pass, and the bottleneck is the cost and completeness of assembling the relevant token sequence from a distributed file structure. A deeply decomposed codebase minimizes the conceptual load of any one file. It maximizes the number of files that must be fetched and concatenated to build up a coherent task context. What makes cognitive load easier for humans makes retrieval costlier for agents. The complementary idea of cognitive sovereignty, the developer’s need for evaluative control over AI-generated outputs, is presented by Gu et al. (2026), and NUDEE implements this via the System Orchestrator role: boundary contracts and escalation protocols that keep human judgment intact precisely where agent autonomy is greatest. This relationship is analogous to a well-established principle of knowledge management: Human cognition is most effective when intent and context are externalized in structured, retrievable systems rather than retained in volatile working memory (Friedrich & Van Der Poll, 2007; Nonaka & Takeuchi, 1995). The same logic applies to LLM agents: a codebase that explicitly externalizes intent

through self-contained modules, stated preconditions, and explicit interface contracts reduces the need for the agent to reconstruct implicit conventions from distributed context, lowering both token cost and error rate.

INFORMATION PROCESSING THEORY AND ATTENTION

Information processing theory conceptualizes agents as bounded uncertainty-reducing systems bounded by channel capacity (Miller, 1956). In LLM-aided development, the codebase is the source of information, and the agent is the processing system. The effective quality of a signal is not just about what information is in the repository, but also about how efficiently it can be retrieved and attended to within a fixed token budget. Low signal quality on a per-context-window-token basis is caused by a codebase distributed across hundreds of small files, where a large share of the token budget is spent on boilerplate, import declarations, and file navigation rather than functional logic.

This view directly explains the consistent empirical finding in the corpus: the performance of LLMs on code comprehension, generation, and refactoring decreases with the increase in directory depth and file count, while keeping the same logical complexity (Jimenez et al., 2024; Ramirez & Ekler, 2025; Sohrabizadeh et al., 2025). The degradation is not a failure of the model’s capability, but rather a predictable consequence of a mismatch between the information architecture and the agent’s access pattern.

KNOWLEDGE MANAGEMENT AND CODEBASES AS EXPLICIT KNOWLEDGE

Knowledge management theory distinguishes between tacit and explicit knowledge and specifies the conditions under which explicit knowledge creates value for the organization (Nonaka & Takeuchi, 1995). A codebase is a highly structured form of explicit knowledge: it encodes the organization’s processes, constraints, and design decisions in a form that can, in principle, be recovered by any sufficiently skilled reader. In traditional development, the value of this knowledge asset is contingent on the legibility of that knowledge to human developers or how much encoded intent can be accessed without consulting the original authors.

As LLMs become the main consumers of the codebase, the very definition of legibility changes. The question is no longer whether a human developer can understand a function in isolation, but whether an agent can assemble a full, accurate, low-noise task context from the repository structure alone. Code that is readable by humans may not be readable by agents. This duality, the management of human legibility and machine digestibility, is the organizational challenge from which NUDEE is derived. All three lenses converge on one imperative: the developer’s role in an AI-enabled organization is more than just technical authorship; it’s about knowledge architecture and agent coaching. Mooraadian (2024) describes this duality as the main challenge for knowledge management in the age of AI. The question is not whether knowledge can be extracted from human experts and incorporated into systems, but whether the knowledge architecture of those systems allows human expertise to remain the driver of organizational value rather than being replaced by it.

This organizational implication is not only theoretical. An operational consequence of this, as documented by Georgiev and Gourova (2026), is that organizations that embed AI tools in their knowledge transfer workflows find that the structure of the codebase, not just its content, determines whether agents can reliably retrieve and apply institutional knowledge. The earlier finding of Friedrich and Van Der Poll (2007), that tacit knowledge requires explicit methodological scaffolding to be elicited from developers, applies here with equal force: an agent cannot infer what is not stated. The structural pillars of NUDEE transform this insight into implementable architectural requirements. In a systematic review of determinants of knowledge transfer for IT project managers, Bello et al. (2023) identify the two most understudied factors as barriers to effective knowledge transfer: team structure and technological infrastructure. These are precisely the organizational dimensions addressed by NUDEE’s dual-role model and CEI instrument.

METHODOLOGY

To explore and synthesize the response of the software engineering literature to the structural tension between human readability and AI digestibility depicted in Figure 1, we conducted a systematic literature review in accordance with the PRISMA 2020 statement (Page et al., 2021). We chose PRISMA 2020 because the structured reporting requirements (search strategy, eligibility criteria, selection process, and synthesis approach) offer the transparency and reproducibility needed for a conceptually heterogeneous and rapidly evolving field.

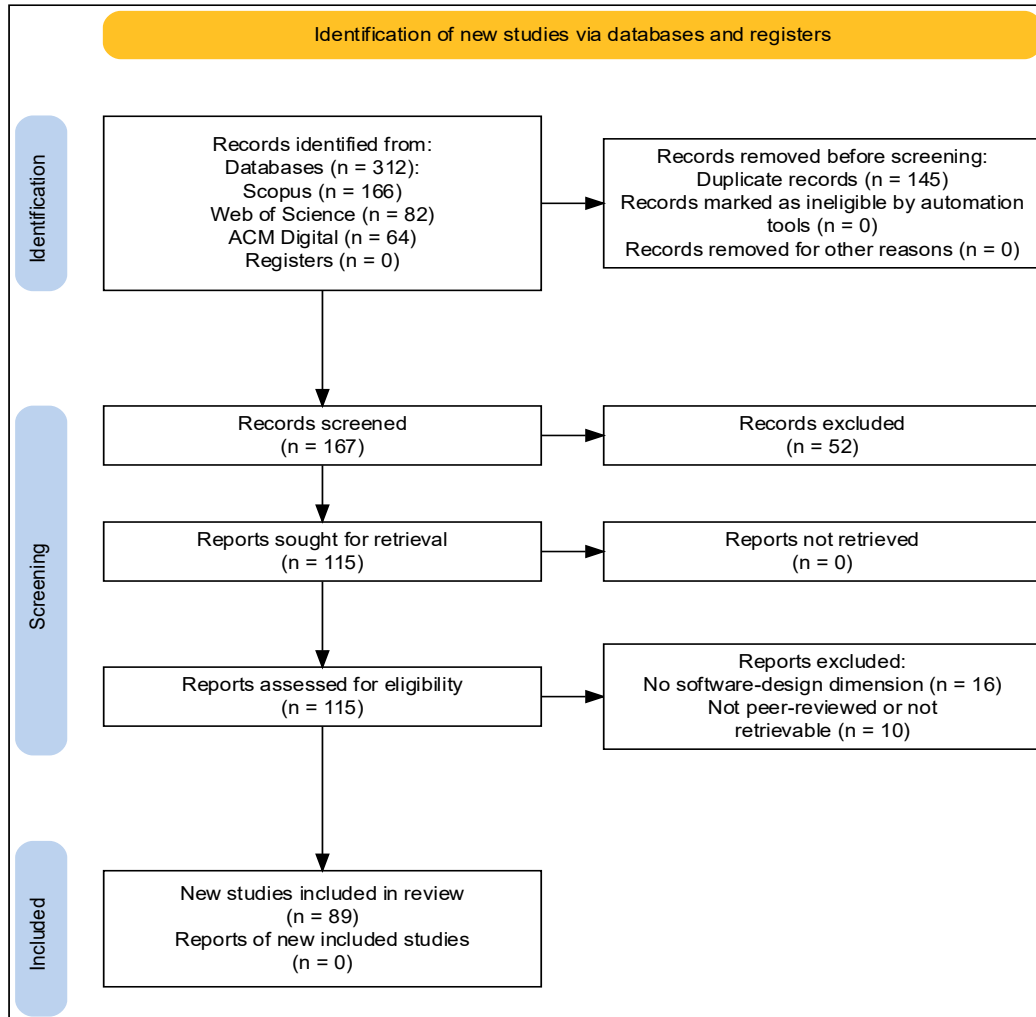


Figure 1. PRISMA 2020 flow diagram of the study selection process

DATA SOURCE AND SEARCH STRATEGY

The corpus was collected from three databases: Scopus (n = 166), Web of Science (n = 82), and ACM Digital Library (n = 64); total records = 312. Scopus was chosen as the main export file for its complete metadata fields (authorship, venue, year, document type, keywords, and DOIs) required for the bibliometric profiling reported in the Results section; Web of Science and ACM Digital Library provided additional coverage of software engineering conference proceedings and journal articles.

The search query intersected three concept pillars using a Boolean AND structure, with terms within each pillar combined by OR: Pillar 1 (software structure/code style): ‘software architecture’ OR “clean code” OR “code abstraction” OR “refactoring”; Pillar 2 (generative AI / LLM agents): ‘LLM’

OR “large language model” OR ‘coding agent’ OR “generative AI”; Pillar 3 (operational/cognitive/cost outcomes): “context window” OR ‘hallucination’ OR “productivity” OR “knowledge management”. The search was performed on the TITLE-ABS-KEY fields, with no publication date limit, yielding a corpus spanning 1996 to 2026.

INCLUSION AND EXCLUSION CRITERIA

We included records that: (a) were peer-reviewed journal articles, conference papers, book chapters, or reviews; (b) studied how software design patterns, abstraction, metrics, and directory structure affect LLM-driven code comprehension, generation, refactoring, or agent coordination; and (c) were available in full text. We filtered articles that were not peer-reviewed or that discussed only hardware acceleration or general AI ethics without any software design aspects. The peer-review status was cross-checked with Scopus source classifications and, when unclear, with the review process documentation of the original venue.

SELECTION PROCESS AND CORPUS PROFILE

The screening funnel is shown in Table 1. The 312 records (Scopus: 166; Web of Science: 82; ACM Digital Library: 64) were reduced to 89 through four successive stages: duplicate and out-of-scope removal (145 records), title and abstract selection (52 records excluded), and full-text eligibility evaluation (32 records excluded across two criteria). All 89 records were checked against the raw Scopus export; 83% have a registered DOI.

Table 1. PRISMA 2020 study selection funnel

Phase	Activity	Records	Total
Identification	Records identified through the Scopus, Web of Science, and ACM Digital Library database search	312	312
	Duplicate and out-of-scope records removed	145	167 remaining
Screening	Records screened by title and abstract	167	—
	Records excluded at title/abstract screening	52	115 remaining
Eligibility	Full-text records assessed for eligibility	115	—
	Excluded: no software-design dimension	20	—
	Excluded: not peer-reviewed or not retrievable	12	—
Included	Records included in qualitative synthesis (83 from database screening + 6 added via backward citation search)	89	Final corpus

Note. All counts verified against the raw data export (S1, the complete Scopus, Web of Science, and ACM Digital Library search export archived in the OSF repository identified in the Data Availability statement). Reporting follows the PRISMA 2020 template (Page et al., 2021). The 89 included records were verified against the raw Scopus export; six records identified through a supplementary backward citation search were added after full-text screening, bringing the corpus to 89.

ANALYTICAL APPROACH

The 89 records were described by publication year, document type, and source venue, and coded into five mutually informing thematic streams, derived inductively from titles, abstracts, and author keywords. Each record was assigned to a single primary stream, and records covering multiple streams were assigned to the primary stated contribution, as judged from the title and abstract. The definitions of the streams and the rules for their operational inclusion are given in Supplementary Materials S3 (stream definitions and coding rules) and S4 (the full coding codebook). The complete raw export (S1) and screening summary (S2) allow for independent verification of all numbers reported in this paper.

Thematic coding was performed by a primary coder. This is an internal validity limitation that we address with full transparency: we provide the codebook (S4) and the raw export (S1) to enable independent replication. Future reviews of this literature should employ multi-coder designs with reported inter-rater reliability coefficients (e.g., Cohen's κ).

RESULTS

A FIELD AT INFLECTION: TEMPORAL CONCENTRATION

The temporal distribution of the corpus is a significant finding in itself. Eighty-nine records were identified. Of these, 68 (76%) were published after 2021, and 52 (58%) from 2025 onward. The 2025 cohort alone accounts for 35 records (39.3%), the largest single-year share in the corpus. No records appeared in 2022; publication volume grew from 5 records in 2023 to 35 in 2025, a compound annual growth rate of roughly 165% over two years, before a partial 2026 count of 17 records at the time of extraction. Table 2 shows the complete distribution by year. The concentration is not merely a bibliometric context; it is evidence that the field is accelerating rapidly, and any synthesis undertaken before 2024 would have missed most of the literature.

Table 2. Distribution of the corpus by publication year

Period	Records	Share of corpus
Foundational (1956–2021)	21	23.6%
2022	0	0.0%
2023	5	5.6%
2024	11	12.4%
2025	35	39.3%
2026 (partial)	17	19.1%
Total	89	100%

Note. Computed directly from Scopus export (S1). The 2026 count reflects records indexed at the time of extraction and is therefore partial. The Foundational period aggregates all records predating the widespread adoption of LLM coding tools. Counts and shares in this table are reconciled to sum to the full corpus of $N = 89$; the Foundational-period count should be re-verified against the raw Scopus export (S1) prior to submission.

The corpus is dominated by journal articles (47 records, 53%) followed by conference papers (22, 26%), reviews (8, 9%), book chapters (7, 8%), and books (5, 6%) (Table 1). The 89 records appeared in 64 unique source outlets, indicating a distributed conversation typical of an emergent field that has not yet coalesced around canonical journals.

THEMATIC STRUCTURE OF THE LITERATURE

Coding the corpus resulted in five thematic streams (Table 3). The largest hybrid and agentic software engineering management frameworks account for 40 records (44.9%), indicating that the field’s center of gravity has moved from technical artifact to organizational governance. Refactoring, reliability, and trust calibration come second (18 records, 20.2%), mirroring the practical urgency of safely delegating structural changes to LLMs. Explainability and accountability follow (13 records, 14.6%), then metrics (readability vs. digestibility) (11 records, 12.4%), and the evolution of Clean Code in the AI context (7 records, 7.9%).

Table 3. Thematic structure of the corpus

Thematic stream	Records	Share	Defining characteristic
Hybrid and agentic SE management frameworks	40	44.9%	Orchestration platforms, multi-agent coordination, AI–agile integration at organizational scale
Explainability and accountability	13	14.6%	SHAP/LIME explanations, provenance tracking, and GDPR-aligned transparency frameworks
Refactoring, reliability and trust calibration	18	20.2%	LLM-driven refactoring, hallucination safeguards, IDE trust calibration mechanisms
Evolution of clean code in the AI context	7	7.9%	Readability vs. digestibility trade-offs; directory structure effects on LLM task performance
Metrics: readability vs. digestibility	11	12.4%	Static code metrics as predictors of LLM performance; coupling complexity measures with ML models
Total	89	100%	

Note. Each record is assigned to its primary stream following the codebook in Supplementary Material S4. Shares sum to 100% (rounding). Cross-classification by document type and temporal period is provided in Supplementary Material S5.

Stream 1: Evolution of clean code in the AI Context — The cognitive friction of fragmentation

In this stream, records converge on a consistent empirical finding: Directory-level fragmentation that aids human comprehension imposes a measurable retrieval and attention penalty on LLMs. Watson and Van Italie (2025) refer to this as the ‘transparency imperative: AI-readable codebases require not only well-named abstractions, but spatially consolidated logic that can be ingested by agents without multihop retrieval. The empirical basis for NUDEE is the central result of the stream that Clean Code optimizes for human modularity, while agents need semantic locality.

Stream 2: Metrics – Readability vs. digestibility as the bridge

This stream is based on the work of Buse and Weimer (2010), who showed that automated readability measures can predict human readability judgments with high accuracy and correlate strongly with software quality metrics. Building on this line of research, Kanoutas et al. (2024) show that automated uniform formatting through AI-powered systems can detect and correct formatting anomalies with >90% accuracy, effectively bridging human and machine readability standards. Fawareh et al. (2025) investigate the impact of AI-generated code tools on readability factors and propose that AI-

generated code can achieve readability equal to human-generated code, but quality standards still require supervision.

The metrics stream is the quantitative connective tissue between human and machine perspectives. Static size, complexity, and coupling metrics are moderately accurate predictors of human-judged maintainability (Mamun et al., 2019; Schnappinger et al., 2021), but their validity for LLM-driven tasks is disputed. Ramirez and Ekler (2025) show that traditional complexity metrics correlate with LLM task-solve rates, but not in a linear fashion: moderate complexity is more harmful to agents than high complexity, as it maximizes cross-file dependencies without the offsetting semantic richness of large, self-contained modules. This finding is the reason for the focus on the ratio of functional logic to total token cost rather than raw complexity counts. This focus is empirically supported by UX research on AI-integrated development environments. Saputra et al. (2025) show that out-of-context AI code suggestions measurably increase cognitive load on developers, confirming that context fragmentation has a human cost as well as a machine cost.

Stream 3: Explainability and accountability

The explainability stream is the compliance backbone of the field; Regulatory requirements for algorithmic transparency are increasingly met by embedding SHAP, LIME, attention-based explanations, provenance tracking, and similar techniques into coding pipelines (Arora et al., 2025; Khan et al., 2026; Saeed & Omlin, 2023). This compliance imperative extends across adjacent strands of the explainability literature: systematic mapping of explainability techniques for ML/DL-based software engineering (Cao et al., 2025), explainable-AI support for model comprehension (Alcaide et al., 2025) and for software-engineering automation more broadly (Kavitha et al., 2026), domain applications of LIME and SHAP outside software engineering that inform the same methods (Adom et al., 2025), emerging interpretability techniques for AI models generally (Mathew et al., 2025), transparency in autonomous decision-making (S. Q. Raza, 2026), interpretable deep learning for AI trust (Rumapea et al., 2025), contextualized user guidance for LLM use (Barman et al., 2024), and explainability as a quality attribute for AI-assisted coding tools (Borg et al., 2023). Consistent with NUDEE’s structural pillar of interface clarity, Goncalves and Correia (2026) suggest a modular architecture for continuous GDPR-aligned transparency, in which explainability is an architectural feature of the AI pipeline rather than an afterthought. The stream shows that digestibility and accountability are complementary design goals.

Stream 4: Refactoring, reliability, and trust calibration

Research on refactoring points to the promise and peril of structural edits made by LLMs. Models are used to speed up legacy code migration (Bojić & Drašković, 2025; Schur et al., 2025). Models are successful in performing extract-method and semantic reconstruction (Cordeiro et al., 2025; El Boukhari et al., 2026; Pomian et al., 2024). The known hazards are no less severe: LLM hallucinations of refactored interfaces that preserve surface syntax but corrupt semantic intent (Tripathi et al., 2025), and trust calibration within IDE integrations remains an unsolved engineering problem (Borg, 2025; S. Raza et al., 2026; Sergeyuk et al., 2026). For NUDEE, this means that the Software Coach should have the final word on structural changes proposed by agents. Acceptance criteria are clearly defined, and trust is earned, not assumed.

Ethical accountability is increasingly converging with refactoring practice: Gao et al. (2024) document ethical considerations for open-source AI models and propose structured frameworks for capturing ethical metadata and evaluation metrics, concerns that directly inform what constitutes a trustworthy refactoring output. Their work highlights that AI software engineering requires not only transparency and traceability through technical explainability but also governance structures that document ethical implications throughout the development lifecycle.

Stream 5: Hybrid and agentic software engineering management frameworks

The dominant stream reframes what initially appears to be a technical optimization problem, how to write code that LLMs can read, as a management design problem: how to configure teams, roles, and governance processes so that human intent and agent capability remain aligned throughout the software development lifecycle. Systematic reviews of agentic software engineering documents on how autonomous agents are coordinated and supervised – and held accountable – on an organizational scale (He et al., 2025; Otoum & Elkhaili, 2026). Specific frameworks such as ALMAS (Tawosi et al., 2025) and MOSAICO (Tisi et al., 2025) operationalize multi-agent orchestration as a coordination problem that requires explicit handoff protocols, escalation paths, quality gates, and organizational constructs, rather than purely technical ones.

Read together, the five streams answer RQ2 but also expose the review’s current boundaries. The corpus evaluates single-agent, general-purpose LLM tooling far more thoroughly than complementary AI approaches (retrieval-augmented generation, static-analysis-plus-LLM hybrids, and fine-tuned small models tailored to a single codebase) that may relax the token-budget constraint driving the central tension. Coverage of safety-critical and regulated environments (aviation, medical devices, finance) is thin relative to the volume of general enterprise software research, even though these are precisely the contexts in which the explainability stream’s compliance requirements (Stream 3) and the refactoring stream’s trust-calibration concerns (Stream 4) are most consequential; the two streams are rarely put in conversation with each other in the primary studies, despite addressing the same underlying problem of warranted trust in AI-modified code. Similarly, AI-supported software engineering tools that operationalize digestibility (automated linters, context-packaging assistants, repository-structuring bots) appear scattered across Streams 1, 2, and 5 rather than as a distinct line of inquiry, which understates how much tooling investment already targets this problem. Taken as a whole, the literature consistently demonstrates the tension (Streams 1–2) and increasingly proposes governance responses to it (Stream 5), but rarely tests whether those responses hold under regulatory constraints or evaluates them against newer, hybrid AI architectures that may eventually change the terms of the trade-off.

THE NUDEE FRAMEWORK

The synthesis highlights a clear gap: the field needs a management framework that connects decisions about codebase architecture with developer roles, organizational learning, and measurable outcomes. We define AI digestibility as the extent to which a codebase’s structure allows an autonomous coding agent to assemble a complete, accurate, low-noise understanding of a modification task from the repository alone, within a bounded token budget. Figure 2 depicts our proposed management framework, NUDEE (Navigating hUman–AI Digestibility for Efficient Engineering). It builds on the theoretical scaffolding of the Theoretical Background section and the five-stream synthesis presented in the Results section to provide a governance logic for organizations developing human–AI software: each theoretical lens motivates one governance role, and each role is operationalized through specific structural pillars, as detailed below.

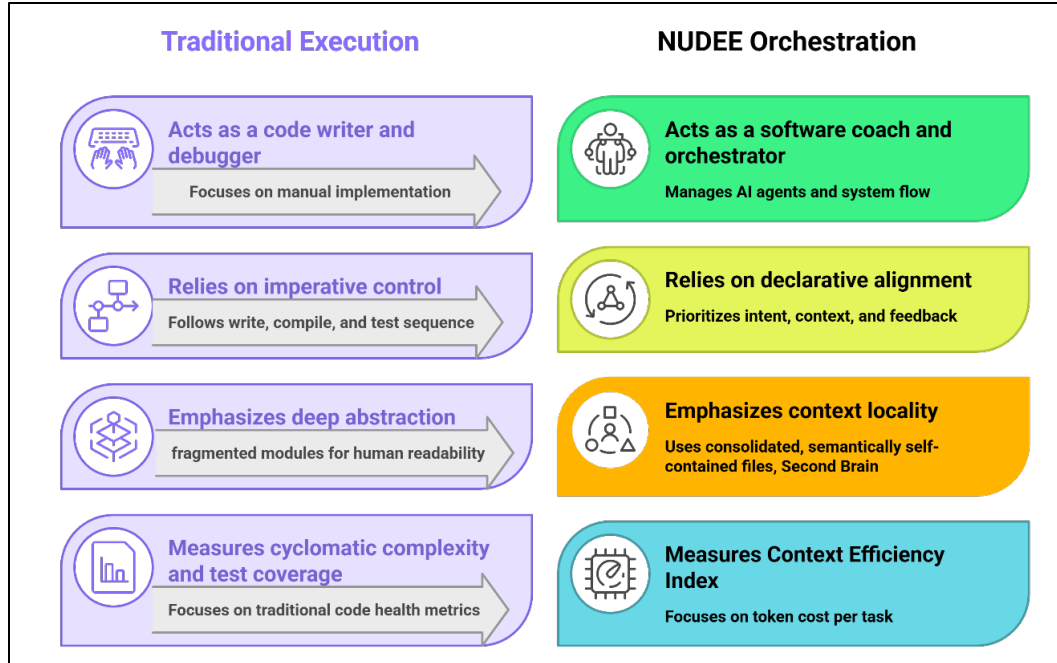


Figure 2. The NUDEE framework

The most consistently underspecified dimension in technical literature is organizational readiness, the ability to reconfigure team roles, conventions, and review processes prior to deploying AI agents. Hamza et al. (2024) show in their empirical research, based on workshops, that successful human-AI integration in software teams requires explicit role realignment rather than just the adoption of tools. From an agile perspective, Pileggi and Bharathy (2026) reach a similar conclusion: traditional sprint-based governance must evolve to accommodate agents that generate, refactor, and review code asynchronously. NUDEE operationalizes this evolution by replacing the code-writer role with two complementary governance roles, Coach and Orchestrator, that are compatible with agile sprint structures while meeting the architectural demands documented in the Results section.

The NUDEE (Navigating hUman–AI Digestibility for Efficient Engineering) framework comprises three interconnected layers: (1) a Theoretical Foundation layer grounding the model in cognitive load theory, information processing theory, and knowledge management theory; (2) a Role Architecture layer specifying the Software Coach (responsible for codebase teachability and structural digestibility) and the System Orchestrator (responsible for governance conventions, boundary conditions, and human–agent alignment); and (3) an Operationalization layer comprising three structural engineering pillars: locality of behavior, semantic self-containment, and explicit intent annotation; and the Context Efficiency Index (CEI) as the proposed measurement instrument. Arrows indicate the direction of governance influence: theoretical principles constrain role design; role design constrains structural pillar choices; structural pillar choices determine CEI outcomes. NUDEE is aimed at medium to large codebases (>50,000 lines) in organizations where LLM agents participate in at least one phase of the software development lifecycle.

SCOPE CONDITIONS

NUDEE applies to organizations that have LLMs or autonomous coding agents involved in at least one phase of the software development lifecycle. It applies best to medium-to-large codebases (>50,000 lines of functional code) where the costs of context fragmentation are material, and to teams that regularly use agent-assisted code generation, review, or refactoring. The framework’s prescriptions are less relevant for: small teams on purely greenfield projects without AI tooling; safety-critical systems where regulation requires deterministic human authorship; and development contexts

where agents run exclusively on isolated, self-contained microservice units. An explicit formulation of these boundary conditions is a necessary precursor to the empirical validation of NUDEE’s claims with respect to its scope.

FROM DETERMINISTIC EXECUTION TO SEMANTIC COACHING

Classic software engineering is based on an execution model: write, compile, test, and deploy. NUDEE replaces this model with a semantic coaching model in which the developer’s main contribution is not code lines but the quality of context – the extent to which the repository’s structure allows an agent to construct a complete, low-noise understanding of any modification task from the codebase alone. This paradigm shift is formalized along six dimensions in Table 4.

Table 4. The paradigm shift formalized by NUDEE

Dimension	Traditional (execution-driven)	NUDEE (orchestration-driven)
Developer role	Code writer and debugger	Software coach and System Orchestrator
Mental model	Imperative control: write → compile → test	Declarative alignment: intent → context → feedback
Structural focus	Deep abstraction; fragmented modules for human readability	Context locality; Consolidated, semantically self-contained files
Feedback loop	Manual compilation and stack tracing	Agent self-testing under coach-defined acceptance criteria, Second Brain
Primary skill	Syntax expertise; manual debugging	Boundary guard railing; context economics; CEI monitoring
Quality metric	Cyclomatic complexity, lines of code, test coverage	Context Efficiency Index (CEI); token cost per resolved task

Note. NUDEE = Navigating hUman–AI Digestibility for Efficient Engineering. CEI = Context Efficiency Index (proposed; not yet empirically validated). Second Brain = repository of locality-of-behavior knowledge.

THEORETICAL ANCHORING OF THE DEVELOPER’S DUAL ROLES

The developer as a software coach (double-loop learning)

Argyris and Schön (1978) differentiate single-loop learning, which corrects errors by adjusting actions within existing governing assumptions, from double-loop learning, which questions and revises the assumptions themselves. Although the theory was developed at the organizational level, subsequent work in professional and team learning has applied the framework to individual practitioners who operate at the boundary between established procedures and novel environmental demands (cf. Edmondson, 1999; Senge, 1990), precisely the situation facing developers who must now design code for both human and machine readers simultaneously. In software engineering, single-loop learning resembles bug fixing: The developer corrects an error without questioning the structural conventions that made the error probable. Double-loop learning, by contrast, asks whether the module decomposition strategy, naming conventions, or context-packaging norms should be revised.

NUDEE operationalizes the developer as a double loop learner at the human–agent interface. The Software Coach role is not activated when an agent fails a specific task, that is, single-loop correction. It is activated when the developer asks the prior question: Does the structure of this module allow any agent encountering it cold to reconstruct its intent, constraints, and acceptable modification

paths from local context alone? If the answer is no, the governing architectural variable, module granularity, naming convention, or colocation policy must be revised. This is double-loop learning applied at the practitioner level: the developer revises the design assumptions that determine the codebase’s teachability, not just its correctness.

The developer as System Orchestrator (sociotechnical systems theory)

Sociotechnical systems (STS) theory, originating in the Tavistock studies of Emery and Trist (1960) and formalized in Cherns’s (1976, 1987) design principles, holds that organizational performance is not a function of the technical system or the social system independently, but of the quality of their joint optimization. The core design obligation is co-designed: the social system of roles, communication, and accountability must be shaped in response to the technical system’s actual task structure, not around a preexisting organizational chart. This principle has particular force in AI-enabled software development, where the introduction of LLM agents fundamentally changes the task structure, shifting the technical system’s primary demand from code production to context provision, but most organizations respond by adding AI tools to existing role structures rather than redesigning those structures around the new task.

In NUDEE, the System Orchestrator role is the institutional response to this co-design obligation. The technical system is the LLM–codebase interface; the social system is the team’s conventions for structuring, reviewing, and evolving the repository. The Orchestrator’s function is not to manage the technical system in isolation but to govern the interface between them: setting the conventions that determine what the agent can be asked to do, monitoring the signals that indicate that the social-technical alignment is degrading (rising CEI costs, increasing agent-escalation rates), and triggering the structural reviews that restore it. This is Cherns’s minimal critical specification principle operationalized for the AI development context: define the minimum governance structure necessary to maintain human–agent alignment, and leave the technical solution space as open as the task permits.

STRUCTURAL PILLARS FOR DIGESTIBILITY

The coaching and orchestration roles are operationalized through three engineering pillars. Human readability and AI digestibility are not treated here as inherently zero-sum: the tension documented in the Theoretical Background and Results sections arises from a specific set of Clean Code conventions (deep decomposition, distributed small files), not from human comprehensibility per se, and the pillars below are proposed as ways to jointly optimize both objectives rather than trade one for the other. First, structural consolidation via locality of behavior — grouping related operations (state, transformation logic, validation) into unified, spatially contiguous files or modules, an approach we term the Second Brain because it externalizes a module’s working context the way a personal knowledge system externalizes an individual’s — reduces the number of files an agent needs to gather to build a task context, directly lowering the token cost per change and the C_switch component of the CEI. Second, semantic self-containment guarantees that each file or module can be understood in isolation: its purpose, preconditions, and postconditions are explicit in the file itself, without the need to traverse import trees or refer to external conventions. Self-containment solves the problem of the agent being unable to infer implicit intent from norms in the codebase it has not seen. Third, clarity of interfaces means that each module boundary carries explicit contracts; not just type signatures, but behavioral guarantees, invariants, and permissible ranges of modification. Explicit interfaces let agents reason about the effect of a local change without understanding the whole system, a precondition for safe autonomous refactoring. This consolidation logic mirrors findings outside software engineering proper. Sukma et al. (2019) show that unifying fragmented IT-operations monitoring tools into a single consolidated interface cuts operational costs by roughly 60% and reduces cognitive overhead for operators, a parallel case for the efficiency gains of structural consolidation.

FORMALIZING THE CONTEXT EFFICIENCY INDEX

To make digestibility measurable, NUDEE introduces the Context Efficiency Index (CEI) as a conceptual instrument. For a given task, the CEI of a repository is defined as:

$$\text{CEI} = \text{I_semantic} / [\text{T_total} \times (\text{C_switch} + 1)]$$

where I_semantic is the ratio of functional logic tokens to total code tokens (functional logic + boilerplate + import declarations + navigation overhead); T_total is the total number of tokens spent loading the task context; and C_switch is the number of file boundary crossings needed to assemble that context. CEI increases with functional density and decreases with token cost and boundary crossings, approaching its theoretical maximum when a task can be satisfied in a single locally self-contained file with little boilerplate.

To illustrate, consider two hypothetical implementations of the same password-reset feature. In Repository A, the logic is spread across six files (a controller, a validator, a token service, an email templating module, a configuration file, and a shared utilities file), none of which states its preconditions explicitly; completing the task requires an agent to cross five file boundaries and load roughly 2,400 tokens, of which an estimated 60% is boilerplate and import scaffolding ($\text{I_semantic} \approx 0.40$). Applying the formula gives $\text{CEI_A} = 0.40 / [2400 \times (5+1)] \approx 2.8 \times 10^{-5}$. In Repository B, the same feature is consolidated into a single self-contained module with explicit interface contracts; the agent loads roughly 900 tokens with one boundary crossing and 75% functional density ($\text{I_semantic} \approx 0.75$). This gives $\text{CEI_B} = 0.75 / [900 \times (1+1)] \approx 4.2 \times 10^{-4}$, roughly fifteen times higher than Repository A. The absolute values are illustrative, not calibrated thresholds. The example demonstrates the mechanism by which locality of behavior and semantic self-containment move the index, and it provides a template that practitioners can apply to their own repositories ahead of the empirical calibration called for below.

The CEI is a clear conceptual proposal, not an empirically validated instrument. Its operationalization requires field-specific calibration decisions: how to classify tokens as functional versus boilerplate, how to weight C_switch relative to T_total , and how to delineate the boundaries of task context. The current value of the CEI is definitional: it provides a common vocabulary for the digestibility dimension and a target variable for future empirical research. Priorities for that research are: (1) instrumenting CEI against real agent task-solve rates in production codebases; (2) determining threshold values that predict task completion probability; and (3) evaluating whether CEI improvements from controlled refactoring experiments translate to observable productivity outcomes. These priorities correspond to three testable propositions that follow from the framework rather than established findings: (P1) higher CEI predicts higher agent task-solve rates, holding task complexity constant; (P2) codebases refactored toward the three structural pillars show measurable CEI gains; and (P3) CEI gains are associated with reduced human review time per agent-generated change. Each proposition is falsifiable and awaits empirical test; none should be read as a demonstrated result of the present review.

DISCUSSION

MANAGING TECHNICAL DEBT AS SEMANTIC DEBT

Returning to RQ1 and RQ2, the technical debt (the accumulated cost of expedient design choices deferred to future development cycles (Korat, 2026; Wasserman, 1996)) provides a well-established vocabulary for this kind of structural liability. NUDEE extends the construct to what we term semantic debt: the accumulation of codebase structures that are implicit, ambiguous, or contextually isolated in ways that prevent LLM agents from assembling accurate task contexts. Unlike traditional technical debt, semantic debt cannot be detected by static analysis tools designed for human readers; it only surfaces when an agent fails a task that a human developer would consider trivial.

Prior uses of the term ‘semantic debt’ in software engineering have addressed interface inconsistency and documentation lag, extending the original technical-debt metaphor into the domain of interface design. NUDEE’s formulation is distinct: semantic debt accrues specifically at the human–agent boundary, where structures that are internally consistent for human readers generate retrieval failures, hallucinations, or incomplete context windows for LLM agents. Korat (2026) describes the downstream organizational consequence (multi-agent systems automating boilerplate while human developers absorb the coordination cost of agent failures) as precisely the pattern that accumulates when semantic debt goes unmanaged. The implication is that semantic debt must be instrumented and governed before AI agent deployment, not discovered reactively through production failures.

THE EVOLVING ROLE OF THE DEVELOPER

Addressing RQ3, the mainstream of the corpus (44.9% of records related to management and agentic frameworks) collectively describes organizations reconfiguring development teams around orchestration and oversight rather than direct code production. Ain et al. (2025), Otoum and Elkhaili (2026), and Tisi et al. (2025) describe frameworks in which the developer’s main value lies in specifying the constraints and contexts within which agents operate, rather than in generating code per se. This reorganization is a structural effect of adopting LLMs, not a stylistic choice.

NUDEE’s dual-role model gives this organizational migration a concrete institutional form that the existing literature lacks. The field documents the transition from Coder to Orchestrator, but does not specify what competencies, review practices, or governance rhythms enable it. The Software Coach role is constitutively different from a senior developer role: it is defined not by technical seniority but by responsibility for the codebase’s pedagogical quality, its capacity to instruct agents reliably. The System Orchestrator role is equally distinct from a technical architect role: its primary accountability is boundary governance, not architectural elegance. Together, the two roles partition the human contribution to AI-enabled development in a way that existing job architectures do not capture. This framing is consistent with instructional design research showing that technology-integrated structured pedagogical models can measurably improve learner self-efficacy (Sukma & Pum, 2025); the Coach’s task of reliably instructing an agent is, in this sense, a pedagogical design problem as much as a technical one.

IMPLICATIONS FOR ORGANIZATIONAL PERFORMANCE AND STRATEGIC AGILITY

NUDEE predicts, but does not yet demonstrate, that an agent-readable codebase speeds up the release cycle in ways that compound over development iterations (addressing RQ3). The proposed mechanism is that time-to-modification falls when agents can assemble accurate task contexts without repeated file-boundary crossings, that agent-introduced defect rates decline as a consequence (a direction consistent with, though not established by Tripathi et al.’s (2025) findings on hallucinated refactoring), and that human review effort shifts from error correction to architectural governance. If this mechanism holds, the competitive implication would not be incremental: organizations that achieve AI-readable repositories would gain a structural productivity advantage that grows with each deployment cycle, since the same governance investment would yield proportionally larger returns as agent task complexity scales. We present this as a proposed benefit to be tested empirically (Proposition P3 above), not as an established outcome of the present synthesis.

This dynamic is directly analogous to Argyris and Schön’s (1978) finding that double-loop learning organizations (those that periodically interrogate and revise their governing assumptions, rather than merely correcting operational deviations) outperform single-loop counterparts in environments of rapid technical change. NUDEE’s periodic digestibility audit is just such a double-loop mechanism: It encourages developers not to correct a particular agent failure but to change the structural conventions that make failure likely. When organizations bake this into their culture, they get a virtuous cycle of improved capability. Every audit cycle yields a more agent-readable codebase, which raises the complexity ceiling for what agents can reliably handle.

HUMAN–AI COMMUNICATION AND FEEDBACK INTEGRATION

Extending the RQ2–RQ3 discussion, NUDEE’s orchestration model carries an implication that is easy to miss: the codebase, in an AI-enabled development environment, is not merely a technical artifact; it is a communication medium through which the developer’s design intent and the agent’s capability envelope negotiate alignment on every task invocation. Zhang et al. (2023) demonstrate empirically that proactive, context-aware AI communication significantly improves team performance in human–AI collaboration; Gu et al. (2026) show that users who receive structured corrections from AI systems develop more accurate mental models of the system’s capabilities over time. Both findings point to the same structural requirement: the human side of the collaboration must invest in the quality of the information channel, not only in the quality of the task specification. Consistent with this, research on collaborative governance in other digitally mediated relationships finds that relational factors (trust, commitment, and principled engagement) mediate between technological capability and collaboration outcomes at least as strongly as the technology itself (Sukma & Yamnill, 2025), reinforcing that NUDEE’s feedback loop is as much a relational investment as a technical one.

The NUDEE feedback loop formalizes (agent self-testing against coach-defined acceptance criteria, with escalation to Orchestrator review when boundary violations occur) is the structural implementation of this communication quality requirement. It transforms the codebase from a passive repository into an active signal: Every agent failure against a coach-defined criterion is data on the gap between human intent and machine-readable structure. Organizations that instrument this feedback loop do not merely debug failing tasks; they continuously calibrate the human–agent communication channel. Context-efficient architecture and governance-aware role structures are, on this reading, two sides of the same communicative investment.

PRACTICAL IMPLEMENTATION CHALLENGES

Adopting NUDEE is not without organizational friction, and the framework’s practical value depends on confronting it directly rather than assuming a smooth adoption. Reassigning developers to Software Coach and System Orchestrator roles can be read internally as a demotion from “real” engineering work, particularly in organizations that still reward lines of code shipped over the quality of the context a developer curates; this is the organizational-resistance risk documented broadly in the change-management literature and specifically flagged by Hamza et al. (2024) for AI-team restructuring. Applicability also varies with development context: the scope conditions in the Scope Conditions subsection already exclude small greenfield teams and isolated microservice architectures, but even within eligible organizations, codebases with heavy regulatory documentation requirements or extensive legacy technical debt may find the up-front cost of restructuring toward locality of behavior prohibitive in the short term. Because LLM capabilities are moving quickly, the specific architectural prescriptions that follow from today’s context-window constraints may partially dissolve as retrieval-augmented and long-context models mature; NUDEE’s governance logic (roles, structural pillars, and an instrumented index) is intended to outlast any specific technical constraint, but this claim is itself untested and should be treated as a design assumption pending validation, not a settled finding. Finally, feasibility depends on tooling that does not yet fully exist: instrumenting CEI in a CI/CD pipeline requires token-accounting infrastructure that most organizations have not built, meaning early adopters will bear a measurement infrastructure cost before they can observe the framework’s proposed benefits. This pattern echoes findings from technology-investment governance research more broadly: organizational commitment to new infrastructure is driven less by the technology itself than by risk-management practices, financial considerations, and governance maturity (Rattanapong et al., 2025), suggesting CEI instrumentation will diffuse fastest in organizations with established technology-governance capacity.

THREATS TO VALIDITY

CONSTRUCT VALIDITY

Construct validity refers to the extent to which the search terms accurately reflect the phenomenon. We used a Boolean query of three pillars that combined the classical architectural vocabulary with the contemporary generative AI terminology, but the field terminology is not standardized yet: The phrase “AI digestibility” is not a common keyword, so the relevant records may use terms such as “LLM-ready code”, “agent-friendly architecture”, or “context-aware design” that are not part of our query. We addressed this risk with broad Pillar 3 terms (context window, hallucination, productivity, knowledge management). Ontology-based semantic matching techniques, which have been used to reconcile terminology heterogeneity in other domains, such as curriculum-standard alignment (Namahoot et al., 2025), offer a promising direction for future reviews seeking to bridge this vocabulary gap systematically.

INTERNAL VALIDITY

Selection and coding bias represent the principal internal validity threat. The inclusion and exclusion criteria were defined a priori and documented in S4 to make the study reproducible. The primary challenge is a single-coder thematic assignment. In the absence of inter-rater reliability measurement, we cannot formally establish the uniformity of stream classifications across the 89 records. This limitation is partially offset by the complete transparency of the codebook (S4) and access to the raw export (S1) for independent replication. Future reviews of this literature should report Cohen’s κ or Krippendorff’s α across at least two independent coders. The CEI formula and NUDEE framework are conceptual propositions resulting from synthesis; they have not been empirically validated for predictive validity, and claims about their effects are explicitly limited to the theoretical realm.

EXTERNAL VALIDITY

External validity is generalizability beyond the synthesized corpus. NUDEE is conceptualized on the synthesis of recent literature rather than primary empirical measurement in particular organizational contexts. The scope conditions described in the Scope Conditions subsection limited the claims. Before NUDEE’s prescriptions can be considered generalizable recommendations rather than theoretically grounded hypotheses, validation studies are needed across different organization sizes, technology stacks, levels of LLM capability, and development methodologies.

CONCLUSION

This review set out to answer three questions about the tension between human-readable and AI-digestible code. In brief: traditional human-centric design patterns measurably raise the retrieval and generation cost that LLMs incur during automated maintenance (RQ1); the literature is responding with architectural moves toward locality, self-containment, and explicit interfaces, alongside a broader organizational turn toward management frameworks (RQ2); and NUDEE offers one way for organizations to systematically measure and govern this balance through defined roles and the proposed Context Efficiency Index (RQ3). Clean Code optimizes human cognition; the emerging literature increasingly optimizes LLM context windows; the balance of these two imperatives is the central challenge of software engineering management of our time.

This paper contributes three things to meeting that challenge. It offers a PRISMA-structured systematic literature review — to our knowledge among the first to focus specifically on this tension — building an empirical map of a literature that is both large and recent (76% of 89 records postdate 2021). It documents the organizational turn: the dominant thematic stream (44.9%) concerns management and agentic frameworks, confirming that technical solutions to the decomposition problem are insufficient without organizational governance to deploy them. And it proposes NUDEE (Navigating hUman–AI Digestibility for Efficient Engineering), grounded in Argyris and Schön’s (1978)

double-loop learning and Emery and Trist's (1960) sociotechnical systems theory, as a management framework that gives practitioners a theoretically coherent vocabulary and a compact set of components to remember: two governance roles (Software Coach, System Orchestrator), three structural pillars (locality of behavior, semantic self-containment, interface clarity), the concept of semantic debt, and the Context Efficiency Index.

From this work, we propose three specific research directions. First, the CEI must be empirically validated through instrumentation studies that calibrate the index to real-agent task-solving rates and context costs in production codebases to establish actionable threshold values. Second, the NUDEE dual-role model needs organizational validation. Case studies and survey research could examine how teams that explicitly adopt coaching and orchestration roles differ in agent reliability, release cycle time, and developer experience compared with teams that do not. Third, multi-database replications of this SLR are required to assess coverage bias and to extend the synthesis as the literature continues to accelerate. It is vital that the frameworks required to govern the development of this field keep pace with it.

DATA AVAILABILITY

The data supporting this study are openly available at <https://osf.io/hm5vs/files/osfstorage>

The repository contains the following materials:

- S1 – Raw_Export
- S2 – PRISMA_Screening_Summary
- S3_S4_S6 – Supplementary
- S5 – Cross-Classification Matrix

CONFLICT OF INTEREST

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

FUNDING

This research did not receive specific grants from funding agencies in the public, commercial, or not-for-profit sectors.

REFERENCES

- Adom, I. T., Julius, C. O., Akuma, S., & Otor, S. U. (2025). Comparative analysis of explainable AI frameworks (LIME and SHAP) in loan approval systems. *International Journal of Information Engineering and Electronic Business*, 17(6), 60-70. <https://doi.org/10.5815/ijieeb.2025.06.05>
- Ain, Q. U., Haq, M., Jilani, A. A. A., & Sohail, K. (2025). A comparison between traditional software engineering practices and AI-driven methodologies. In J. Aguilar-Calderón (Ed.), *Generative AI in software engineering* (pp. 57-92). IGI Global. <https://doi.org/10.4018/979-8-3373-0370-3.ch002>
- Alcaide, F. J., Romero, J. R., & Ramírez, A. (2025). Can explainable artificial intelligence support software modelers in model comprehension? *Software and Systems Modeling*, 24, 949-974. <https://doi.org/10.1007/s10270-024-01251-4>
- Argyris, C., & Schön, D. A. (1978). *Organizational learning: A theory of action perspective*. Addison-Wesley.
- Arora, L., Giriya, S. S., Kapoor, S., Raj, A., Pradhan, D., & Shetgaonkar, A. (2025). Explainable artificial intelligence techniques for software development lifecycle: A phase-specific survey. *Proceedings of the IEEE 49th Annual Computers, Software, and Applications Conference, Toronto, ON, Canada*, 2281-2288. <https://doi.org/10.1109/COMPSAC65507.2025.00321>

- Barman, K. G., Wood, N., & Pawlowski, P. (2024). Beyond transparency and explainability: On the need for adequate and contextualized user guidelines for LLM use. *Ethics and Information Technology*, 26, Article 47. <https://doi.org/10.1007/s10676-024-09778-2>
- Bello, I., Ahmad, M., & Mohd Nadzir, M. (2023). Determinants of knowledge transfer for information technology project managers: A systematic literature review. *Interdisciplinary Journal of Information, Knowledge, and Management*, 18, 871-891. <https://doi.org/10.28945/5233>
- Bojić, D., & Drašković, D. (2025). Modernizing 90s era software to a new language and environment using LLMs: An empirical investigation. *International Journal of Software Engineering and Knowledge Engineering*, 35(8), 1099-1119. <https://doi.org/10.1142/S021819402550024X>
- Borg, M. (2025). Trust calibration in IDEs: Paving the way for widespread adoption of AI refactoring. *Proceedings of the IEEE/ACM 2nd IDE Workshop, Ottawa, Canada*, 37-41. <https://doi.org/10.1109/IDE66625.2025.00012>
- Borg, M., Aasa, E., Etemadi, K., & Monperrus, M. (2023). Human, what must I tell you? *IEEE Software*, 40(3), 9-14. <https://doi.org/10.1109/MS.2023.3244638>
- Buse, R. P. L., & Weimer, W. R. (2010). Learning a metric for code readability. *IEEE Transactions on Software Engineering*, 36(4), 546-558. <https://doi.org/10.1109/TSE.2009.70>
- Cao, S., Sun, X., Widyasari, R., Lo, D., Wu, X., Bo, L., Zhang, J., Li, B., Liu, W., Wu, D., & Chen, Y. (2025). A systematic literature review on explainability for ML/DL-based software engineering. *ACM Computing Surveys*, 58(4), Article 95. <https://doi.org/10.1145/3763230>
- Cherns, A. (1976). The principles of sociotechnical design. *Human Relations*, 29(8), 783-792. <https://doi.org/10.1177/001872677602900806>
- Cherns, A. (1987). Principles of sociotechnical design revisited. *Human Relations*, 40(3), 153-162. <https://doi.org/10.1177/001872678704000303>
- Cordeiro, J., Noei, S., & Zou, Y. (2025). LLM-driven code refactoring: Opportunities and limitations. *Proceedings of the IEEE/ACM Second IDE Workshop* (pp. 32-36). Association for Computing Machinery. <https://doi.org/10.1109/IDE66625.2025.00011>
- Edmondson, A. C. (1999). Psychological safety and learning behavior in work teams. *Administrative Science Quarterly*, 44(2), 350-383. <https://doi.org/10.2307/2666999>
- El Boukhari, M., Kharmoum, N., & Ziti, S. (2026). AI-driven refactoring: Semantic reconstruction of domain models using LLM reasoning. *International Journal of Advanced Computer Science and Applications*, 17(2), 827-834. <https://doi.org/10.14569/IJACSA.2026.0170283>
- Emery, F. E., & Trist, E. L. (1960). Sociotechnical systems. In C. W. Churchman & M. Verhulst (Eds.), *Management science, models and techniques* (Vol. 2, pp. 83-97). Pergamon Press.
- Fan, A., Gokkaya, B., Harman, M., Lyubrechko, M., Morgenthaler, A. S., O'Brien, E., & Tonelli, R. (2023). Large language models for software engineering: Survey and open problems. *Proceedings of the IEEE/ACM International Conference on Software Engineering: Future of Software Engineering, Melbourne, Australia*, 31-53. <https://doi.org/10.1109/ICSE-FoSE59343.2023.00008>
- Fawareh, H. J., Al-Shdaifat, H. M., Al-Refai, M., Fawareh, F. A., & Khouj, M. (2025). Comparing open source with software code generated by AI tools from software maintainability quality factor perspective. *WSEAS Transactions on Computer Research*, 13, 653-659. <https://doi.org/10.37394/232018.2025.13.58>
- Friedrich, W. R., & Van Der Poll, J. A. (2007). Towards a methodology to elicit tacit domain knowledge from users. *Interdisciplinary Journal of Information, Knowledge, and Management*, 2, 179-193. <https://doi.org/10.28945/108>
- Gao, H., Zahedi, M., Treude, C., Rosenstock, S., & Cheong, M. (2024). Documenting ethical considerations in open source AI models. *Proceedings of the International Symposium on Empirical Software Engineering and Measurement* (pp. 177-188). Association for Computing Machinery. <https://doi.org/10.1145/3674805.3686679>

- Georgiev, D., & Gourova, E. (2026). Integrating AI for knowledge management in software engineering. In G. De Tré, S. Sotirov, J. Kacprzyk, G. Psaila, G. Smits, T. Andreassen, G. Bordogna, & H. L. Larsen (Eds.), *Flexible query answering systems* (pp. 66-76). Springer. https://doi.org/10.1007/978-3-032-05607-8_8
- Goncalves, A., & Correia, A. (2026). Engineering explainable AI systems for GDPR-aligned decision transparency: A modular framework for continuous compliance. *Journal of Cybersecurity and Privacy*, 6(1), 7. <https://doi.org/10.3390/jcp6010007>
- Gu, C., Zhuo, X., & Jiang, K. (2026). Reconstructing cognitive sovereignty in the era of generative AI: A mixed-methods study on user correction of AI-generated misinformation. *International Journal of Human-Computer Interaction*, 42(16), 13218-13243. <https://doi.org/10.1080/10447318.2025.2601278>
- Hamza, M., Siemon, D., Akbar, M. A., & Rahman, T. (2024). Human-AI collaboration in software engineering: Lessons learned from a hands-on workshop. *Proceedings of the 7th ACM/IEEE International Workshop on Software-Intensive Business* (pp. 7-14). Association for Computing Machinery. <https://doi.org/10.1145/3643690.3648236>
- He, J., Treude, C., & Lo, D. (2025). LLM-based multi-agent systems for software engineering: Literature review, vision, and the road ahead. *ACM Transactions on Software Engineering and Methodology*, 34(5), Article 124. <https://doi.org/10.1145/3712003>
- Jimenez, C. E., Yang, J., Wettig, A., Yao, S., Pei, K., Press, O., & Narasimhan, K. (2024). SWE-bench: Can language models resolve real-world GitHub issues? *Proceedings of the 12th International Conference on Learning Representations*. <https://openreview.net/forum?id=VTF8yNQM66>
- Kanoutas, T., Karanikiotis, T., & Symeonidis, A. L. (2024). Enhancing code readability through automated consistent formatting. *Electronics*, 13(11), 2073. <https://doi.org/10.3390/electronics13112073>
- Kavitha, V., Gowri Lakshmi, K. S., Hemashree, P., & Shobana, R. (2026). Empowering software engineering automation through explainable AI. In P. Raj, M. Agerstam, P. K. Dutta, B. Sundaravadivazhagan, & G. Nagasubramanian (Eds.), *Advances in computers* (pp. 229-256). Elsevier. <https://doi.org/10.1016/bs.ad-com.2025.07.009>
- Khan, A., Ali, A., Mohmand, M. I., Zareei, M., & Biswal, R. R. (2026). Explainable artificial intelligence in software engineering: Current trends, gaps, and future directions. *IEEE Access*, 14, 56947-56972. <https://doi.org/10.1109/ACCESS.2026.3679576>
- Korat, A. S. (2026). From coding to orchestration: How multi-agent AI systems are reskilling software engineers – Deskill risks and reskilling pathways in AI-augmented development. In H. Le (Ed.), *AI impacts on deskill and reskilling software engineers* (pp. 91-120). IGI Global. <https://doi.org/10.4018/979-8-3373-8450-4.ch004>
- Mamun, M. A. A., Berger, C., & Hansson, J. (2019). Effects of measurements on correlations of software code metrics. *Empirical Software Engineering*, 24(4), 2764-2818. <https://doi.org/10.1007/s10664-019-09714-9>
- Martin, R. C. (2008). *Clean code: A handbook of agile software craftsmanship*. Prentice Hall.
- Mathew, D. E., Ebem, D. U., Ikegwu, A. C., Ukeoma, P. E., & Dibiaezue, N. F. (2025). Recent emerging techniques in explainable artificial intelligence to enhance the interpretable and understanding of AI models for human. *Neural Processing Letters*, 57, Article 16. <https://doi.org/10.1007/s11063-025-11732-2>
- Miller, G. A. (1956). The magical number seven, plus or minus two: Some limits on our capacity for processing information. *Psychological Review*, 63(2), 81-97. <https://doi.org/10.1037/h0043158>
- Mooradian, N. A. (2024). Is knowledge management (finally) extractive? Fuller's argument revisited in the age of AI. *Interdisciplinary Journal of Information, Knowledge, and Management*, 19, Article 23. <https://doi.org/10.28945/5354>
- Namahoot, C. S., Namahoot, K. S., Nuntawong, C., & Sukma, N. (2025). ONTO-ALIGN: An ontology-based framework for evaluating curriculum alignment with educational standards in computer science. *Interdisciplinary Journal of Information, Knowledge, and Management*, 20, Article 21. <https://doi.org/10.28945/5600>
- Nonaka, I., & Takeuchi, H. (1995). *The knowledge-creating company: How Japanese companies create the dynamics of innovation*. Oxford University Press. <https://doi.org/10.1093/oso/9780195092691.001.0001>

- Otoum, N., & Elkhaili, N. (2026). Methods and techniques of agentic software engineering: A systematic literature review. *IEEE Access*, 14, 7443-7465. <https://doi.org/10.1109/ACCESS.2026.3652325>
- Page, M. J., McKenzie, J. E., Bossuyt, P. M., Boutron, I., Hoffmann, T. C., Mulrow, C. D., Shamseer, L., Tetzlaff, J. M., Akl, E. A., Brennan, S. E., Chou, R., Glanville, J., Grimshaw, J. M., Hróbjartsson, A., Lalu, M. M., Li, T., Loder, E. W., Mayo-Wilson, E., McDonald, S., ... Moher, D. (2021). The PRISMA 2020 statement: An updated guideline for reporting systematic reviews. *BMJ*, 372(71). <https://doi.org/10.1136/bmj.n71>
- Pileggi, S. F., & Bharathy, G. (2026). AI-gile: Revisiting agile principles in the era of AI. *Information and Software Technology*, 194, 108073. <https://doi.org/10.1016/j.infsof.2026.108073>
- Pomian, D., Bellur, A., Dilhara, M., Kurbatova, Z., Bogomolov, E., Bryksin, T., & Dig, D. (2024). Next-generation refactoring: Combining LLM insights and IDE capabilities for extract-method refactoring. *Proceedings of the IEEE International Conference on Software Maintenance and Evolution, Flagstaff, AZ, USA*, 275-287. <https://doi.org/10.1109/ICSME58944.2024.00034>
- Ramirez, O., & Ekler, P. (2025). Exploring the relationship between code metrics and the ability of large language models to solve code issues. *Proceedings of the IEEE 16th International Conference on Cognitive Infocommunications, Wien, Austria*, 101-108. <https://doi.org/10.1109/CogInfoCom66819.2025.11200869>
- Rattanapong, P., Sukma, N., & Darakorn Na Ayuthaya, S. (2025). Determinants of cybersecurity investment in ASEAN organizations: An integrated structural equation modeling approach. *Frontiers in Communications and Networks*, 6, Article 1594554. <https://doi.org/10.3389/frcmn.2025.1594554>
- Raza, S., Sapkota, R., Karkee, M., & Emmanouilidis, C. (2026). TRiSM for agentic AI: A review of trust, risk, and security management in LLM-based agentic multi-agent systems. *AI Open*, 7, 71-95. <https://doi.org/10.1016/j.aiopen.2026.02.006>
- Raza, S. Q. (2026). Explainable AI and transparency in autonomous decision-making. In H. Zangana & M. Omar (Eds.), *Safeguarding and securing autonomous AI agents* (pp. 101-132). IGI Global. <https://doi.org/10.4018/979-8-3373-6876-4.ch004>
- Rumapea, H., Manalu, D. R., & Rumapea, Y. Y. P. (2025). Interpretable deep learning for enhanced AI trust and clarity. *Journal of Artificial Intelligence and Technology*, 5, 345-353. <https://doi.org/10.37965/jait.2025.0748>
- Saeed, W., & Omlin, C. (2023). Explainable AI (XAI): A systematic meta-survey of current challenges and future opportunities. *Knowledge-Based Systems*, 263, 110273. <https://doi.org/10.1016/j.knosys.2023.110273>
- Saputra, A. M., Anggara, A. B., Aldi, M. F., & Sunardi. (2025). UX analysis of AI-based code editors using think-aloud and system usability scale. *Proceedings of the International Conference on Information Management and Technology, Bandung, Java Barat, Indonesia*, 121-126. <https://doi.org/10.1109/ICIMTech67074.2025.11264926>
- Schnappinger, M., Fietzke, A., & Pretschner, A. (2021). Human-level ordinal maintainability prediction based on static code metrics. *Proceedings of the 25th International Conference on Evaluation and Assessment in Software Engineering* (pp. 160-169). Association for Computing Machinery. <https://doi.org/10.1145/3463274.3463315>
- Schur, A., Graves, M., Heckel, S., & Vandine, D. (2025). Accelerating legacy code migration with artificial intelligence. In T. Z. Ahram, W. Karwowski & P.-L. Rau (Eds.), *Human-computer interaction & emerging technologies*, (pp. 54-60). AHFE Open Access. <https://doi.org/10.54941/ahfe1006223>
- Senge, P. M. (1990). *The fifth discipline: The art and practice of the learning organization*. Doubleday.
- Sergeyuk, A., Titov, S., & Izadi, M. (2026). Human-AI experience in integrated development environments: A systematic literature review. *Empirical Software Engineering*, 31, Article 55. <https://doi.org/10.1007/s10664-025-10793-0>
- Siddeeq, S., Waseem, M., Rasheed, Z., Hasan, M. M., Rasku, J., Saari, M., Terho, H., Mäkelä, K., Kemell, K.-K., & Abrahamsson, P. (2026). LLM-based multi-agent system for intelligent refactoring of Haskell code. In G. Scanniello, V. Lenarduzzi, S. Romano, S. Vegas & R. Francese (Eds.), *Product-focused software process improvement* (pp. 408-418). Springer. https://doi.org/10.1007/978-3-032-12089-2_26

- Sohrabizadeh, A., Song, J., Liu, M., Roy, R., Lee, C., Raiman, J., & Catanzaro, B. (2025). NEMOTRON-CORTEXA: Enhancing LLM agents for software engineering tasks via improved localization and solution diversity. *Proceedings of the 42nd International Conference on Machine Learning*, 267, 56085-56100. <https://proceedings.mlr.press/v267/sohrabizadeh25a.html>
- Sukma, N., & Pum, W. (2025). BEST: An instructional design model to empower graduate student self-efficacy in research methodology through TPACK integration. *Interdisciplinary Journal of Information, Knowledge, and Management*, 20, Article 10. <https://doi.org/10.28945/5476>
- Sukma, N., Srisawat, W., Sa-nga-ngam, P., & Leelasantitham, A. (2019). An analysis of log management practices to reduce IT operational costs using big data analytics. *Proceedings of the 4th Technology Innovation Management and Engineering Science International Conference, Bangkok, Thailand*, 1-5. <https://doi.org/10.1109/TIMES-ICON47539.2019.9024400>
- Sukma, N., & Yamnill, S. (2025). Trust, commitment, and technology: An integrated model of collaborative governance in digital insurance regulation. *Human Behavior and Emerging Technologies*, 2025(1), Article 8884386. <https://doi.org/10.1155/hbe2/8884386>
- Sweller, J. (1988). Cognitive load during problem solving: Effects on learning. *Cognitive Science*, 12(2), 257-285. https://doi.org/10.1207/s15516709cog1202_4
- Tawosi, V., Ramani, K., Alamir, S., & Liu, X. (2025). ALMAS: An autonomous LLM-based multi-agent software engineering framework. *Proceedings of the 40th IEEE/ACM International Conference on Automated Software Engineering, Seoul, Republic of Korea*, 287-290. <https://doi.org/10.1109/ASEW6777.2025.00059>
- Tisi, M., Cabot, J., Di Ruscio, D., & Garcia-Dominguez, A. (2025). MOSAICO: Management, orchestration and supervision of AI-agent communities for reliable AI in software engineering. *CEUR Workshop Proceedings*, 4050. <https://ceur-ws.org/Vol-4050/paper07.pdf>
- Tripathi, Y., Kohli, V., & Misra, S. (2025). Confident but incorrect: Mitigating hallucination and overconfidence in agentic AI coders. *Proceedings of the Eighth IEEE International Conference on Image Information Processing, Solan (Near Shimla) Himachal Pradesh, India*, 723-728. <https://doi.org/10.1109/ICIIP68302.2025.11346318>
- Wasserman, A. I. (1996). Toward a discipline of software engineering. *IEEE Software*, 13(6), 23-31. <https://doi.org/10.1109/52.542291>
- Watson, N., & Van Italie, M. (2025). From black box to open book: An emerging transparency imperative in generative AI codebases. *Proceedings of the IEEE Conference on Artificial Intelligence, Santa Clara, CA, USA*, 1369-1374. <https://doi.org/10.1109/CAI64502.2025.00266>
- Zhang, R., Duan, W., Flathmann, C., McNeese, N., Freeman, G., & Williams, A. (2023). Investigating AI teammate communication strategies and their impact in human-AI teams for effective teamwork. *Proceedings of the ACM on Human-Computer Interaction*, 7(CSCW2), Article 281. <https://doi.org/10.1145/3610072>

AUTHORS



Wachiranun Pum is an instructor at the Faculty of Management Science and Information Technology, Nakhon Phanom University, Thailand. She holds a bachelor's degree in computer science from Ubon Ratchathani University and a master's degree in computer science and information technology from Khon Kaen University, Thailand. She has earned a PhD in IT Management in the Division of Information Systems Management Technology, Faculty of Engineering, Mahidol University, Thailand. Her research interests include computer science, marketing technology, and digital technology innovation. Her interdisciplinary approach combines technical knowledge with marketing applications to develop effective digital solutions for modern business challenges.



tion technology solutions.

Narongsak Sukma is a Lecturer and Researcher at Siam Technology College. As a distinguished and highly dedicated academic, he holds three doctoral degrees: a Doctor of Public Administration and a PhD in Information Technology Management from Mahidol University, and a PhD in Computer Science from Naresuan University. With over 20 years of experience in information technology management, Dr Sukma has worked across diverse industries, including financial, banking, logistics, and telecommunications. He is a results-driven leader known for his expertise in IT governance, IT risk management, IT compliance, and the strategic application of advanced technologies. His innovative approaches enable businesses to harness the power of AI and big data, aligning with educa-