

Beyond the ‘Diff’: Addressing Agentic Entropy in Agentic Software Development

MATTEO CASSERINI*, SUPSI, Switzerland

ALESSANDRO FACCHINI, SUPSI, IDSIA, Switzerland and Kozminski University, Poland

ANDREA FERRARIO, SUPSI, IDSIA, University of Zurich, and ETH Zurich, Switzerland

As autonomous coding agents become deeply embedded in software development workflows, their high operational velocity introduces a critical oversight challenge in the form of accumulating divergence between agentic actions and architectural intent. We term this process *agentic entropy*, a systemic drift that traditional code diff-based and HCXAI methods fail to capture, as they address local outputs rather than global agentic behaviour. To close this gap, we propose a *process-oriented explainability* framework that exposes how agentic decisions unfold across time, tool calls, and architectural boundaries. Built around three pillars (conformity seeding, reasoning monitoring, and a causal graph interface) our approach provides intent-level telemetry that complements, rather than replaces, existing review practices. We demonstrate its relevance across two user profiles: lay users engaged in vibe coding, who gain structural visibility otherwise masked by functional success; and professional developers, who gain richer contextual grounding for code review without increased overhead. By treating cognitive drift as a first-class concern alongside code quality, our framework supports the minimum level of human comprehension required for agentic oversight to remain substantive.

CCS Concepts: • **Software and its engineering** → **Software verification and validation**; • **Computing methodologies** → **Intelligent agents**; **Knowledge representation and reasoning**; • **Human-centered computing** → **HCI theory, concepts and models**.

Additional Key Words and Phrases: Agentic Entropy, Process-Oriented Explainability, Human-Centered XAI, Agentic Software Development, Cognitive Debt, Reasoning Traces, Vibe Coding

ACM Reference Format:

Matteo Casserini, Alessandro Facchini, and Andrea Ferrario. 2026. Beyond the ‘Diff’: Addressing Agentic Entropy in Agentic Software Development. 1, 1 (April 2026), 7 pages. <https://doi.org/10.1145/nnnnnnn.nnnnnnn>

1 Introduction

The software development lifecycle is currently undergoing an artificial intelligence (AI) agentic revolution [14]. The transition from large-language-model-based autocomplete features, which provide localized code suggestions, to AI agents represents a fundamental change in the human–AI interaction model. Modern agentic workflows, exemplified by platforms such as Claude Code, can navigate file systems, execute commands, interpret compiler errors, and perform multi-file refactoring with high autonomy [9]. While traditional code diff-based review remains indispensable for

*Corresponding author

Authors’ Contact Information: Matteo Casserini, Department of Innovative Technologies, SUPSI, Lugano, Switzerland, matteo.casserini@supsi.ch; Alessandro Facchini, SUPSI, IDSIA, Lugano, Switzerland, Management in Networked and Digital Societies (MINDS) Department and Kozminski University, Warsaw, Poland, alessandro.facchini@supsi.ch; Andrea Ferrario, SUPSI, IDSIA, Lugano, Institute of Biomedical Ethics and History of Medicine and University of Zurich, Zurich and ETH Zurich, Zurich, Switzerland, aferrario@ethz.ch.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2026 Copyright held by the owner/author(s). Publication rights licensed to ACM.

Manuscript submitted to ACM

inspecting local modifications, it reveals only the outcome of agentic activity. The planning steps, tool-call sequences, cross-file decisions, and contextual inferences that guide an agent’s actions remain largely hidden from human supervisors. We refer to this opacity as **agentic entropy**, a process-level drift whereby autonomous updates optimize for local correctness while eroding global design intent. Agentic entropy unfolds across runs and environments as agents repeatedly operate with limited systemic awareness. A visible manifestation of this drift is **agentic technical debt**, the accumulated structural misalignments, duplicated logic, and fragile refactoring that emerge when entropy goes unchecked.

Entropy accumulation has many causes, including prompt sprawl and stochastic effects. Particularly relevant is the democratization of software development through what is commonly referred to as “vibe coding”, a paradigm where users steer systems with loosely specified prompts and short iterative trial-fix loops, rather than formal code verification and review¹. Functional checks (e.g., confirming that the code runs) can thus mask deeper architectural violations and security gaps, producing an illusion of rapid progress while drift accumulates beneath the surface [3]. Beyond agent behaviour, developers who rely on agentic tools can lose the system-level mental models needed to notice violations, accumulating *cognitive debt* that resides in people rather than code [12]. This co-evolution of agentic entropy and cognitive debt narrows the capability-comprehension gap for oversight, leaving humans procedurally in the loop yet progressively less able to govern the system [15].

To address this gap, we introduce **process-oriented explainability**, a transparency-by-design framework that augments (rather than replaces) existing HCXAI and code-diff practices, helping mitigate agentic entropy. Our framework anchors agent behaviour to machine-readable architectural seeds, monitors intermediate reasoning and tool use, and summarizes activity in a causal reasoning graph for lightweight, global oversight. Its benefits apply to (1) lay users engaged in vibe coding who gain high epistemic safety, as process-level monitoring surfaces structural drift otherwise hidden behind functional success; and (2) professional coder-agent-reviewer workflows who obtain a medium epistemic gain by contextualizing local code diffs within agent-level plans, improving control over agentic execution without increasing review burden.

Our work is structured as follows. Section 2 reviews relevant literature. Section 3 characterizes how agentic entropy manifests in autonomous software development workflows. Section 4 presents our framework and a technical implementation. Finally, Section 5 offers concluding remarks and outlines directions for future research.

2 Related Work

The supervision of autonomous coding agents intersects several established research areas. A foundational pillar is the concept of *software entropy*, first formalized by Lehman in his laws of software evolution, which posits that software systems naturally drift toward increased complexity unless effort is invested in maintaining their structure [13]. Research on human–AI collaboration in software development further highlights how AI-generated code reshapes verification practices. Empirical studies show that reviewers frequently experience cognitive overload or ‘verification fatigue’ when faced with high volumes of agent-produced changes [2, 5]. Within the broader literature on explainable and agentic AI, several frameworks emphasize the need for transparency that captures temporal and sequential structure rather than isolated outputs. Amir [1] argues that explanations for sequential decision-making systems must be trajectory-aware, while the *LEx* framework of Singh et al. [17] emphasizes that explanations should be tailored to stakeholder roles, such as software architects who require higher-level design justifications. Recent advances reinforce this need, as

¹Industry reports show that nearly 45% of AI-generated code fails basic security checks when users rely on prompt-driven workflows without guardrails: <https://www.veracode.com/blog/genai-code-security-report/>

Chain-of-Thought (CoT) prompting surfaces intermediate reasoning [18], and DeepSeek-R1 shows that such traces can be explicitly incentivized during training [7]. Parallel work on agentic AI governance and risk management stresses the need for mechanisms that keep autonomous systems aligned with organizational and architectural constraints [11, 16]. Complementing this, emerging research on agentic software development calls for new forms of transparency and oversight to address non-deterministic tool-use patterns and the high velocity of autonomous coding workflows [8].

3 From Agentic Entropy to Agentic Technical Debt

The emergence of autonomous coding agents requires a reconsideration of how software systems degrade over time. Classical software evolution theory, particularly Lehman’s notion of software entropy, observes that systems naturally drift toward increased complexity unless active effort is invested in maintaining their structure. We extend this perspective to the agentic era by distinguishing **agentic entropy**, which is the ongoing process by which autonomous updates diverge from architectural intent, from **agentic technical debt**, which refers to the accumulated structural misalignments that entropy leaves behind. While Lehman’s entropy arises primarily from human-driven system evolution, agentic entropy emerges from high-velocity, context-limited agent actions that compound across runs and environments. These concepts, while distinct in nature, interact in a reinforcing cycle. Agentic entropy, a *process*, produces agentic technical debt, an *outcome* that manifests in the codebase. The growing opacity of that debt in turn deepens a third, *human-side effect*, namely **cognitive debt** [12], the progressive erosion of the developer’s or reviewer’s system-level mental model as autonomous actions outpace comprehension. As cognitive debt accumulates, the human supervisor’s capacity to detect subsequent entropy diminishes, closing a feedback loop in which undetected drift begets further structural misalignment. In the remainder of this section, we show how this cycle initiates at the process level through three characteristic failure modes in agentic workflows.

The first manifestation of agentic entropy arises from agents optimizing for *local correctness* while drifting from *global architectural intent*. Because agentic workflows, such as those implemented in Claude Code, frequently operate within a constrained context window, they are prone to achieving functional correctness at the module level while inadvertently violating systemic design patterns. As shown by Kabir et al. [10], agents often propose ‘textbook’ solutions that appear locally elegant but overlook system-specific architectural and security constraints. Such mismatches accelerate agentic entropy, introducing redundant or misaligned logic that gradually fragments the codebase’s intended architectural structure [6].

A second manifestation of agentic entropy is the erosion of *semantic stability*, particularly when agents refactor legacy logic without understanding its historical or operational rationale. Autonomous agents excel at identifying and cleaning complex legacy code that appears aesthetically deficient, yet legacy systems often contain ‘ugly’ logic (such as delay loops or seemingly redundant checks) that accounts for undocumented quirks or rare race conditions. When an agent refactors these segments without a deep semantic understanding of their historical necessity, it introduces a structural fragility. The resulting code may pass standardized unit tests but fail in production under specific edge cases that the agent, and subsequently the overwhelmed human reviewer, failed to anticipate. This risk is empirically supported by the work of Barke et al. [2], which characterizes the loss of global context in AI-augmented programming, and Du et al. [4], which demonstrates that while AI models often achieve high functional accuracy, they frequently fail to satisfy structural and architectural constraints. Consequently, the resulting codebase suffers from a ‘stability gap’, where locally functional code lacks the systemic robustness required for long-term evolution.

A third entropy-accelerating factor is the *reviewer’s paradox*, where rising agentic output overwhelms human verification capacity. As autonomous tools accelerate the ‘inner loop’ of code production, the human capacity for

rigorous oversight remains a fixed constraint. This imbalance leads to a breakdown in the traditional code review process, where human auditors move from strategic supervision to passive rubber-stamping. Empirical data suggests that this acceleration does not currently correlate with a commensurate increase in review quality; rather, it often leads to a higher rate of code churn and duplicated logic, further accelerating the system’s entropy [6].

Taken together, these lenses show how agentic entropy converts autonomous decisions into long-term architectural liabilities. Each lens describes a pathway through which entropy accumulates as technical debt unless a supervisory mechanism intervenes. This motivates our shift in the next section from inspecting code outputs to monitoring the entropy-producing processes themselves.

4 Our Approach: Controlling Agentic Entropy with Process-Oriented Explainability

To address the structural risks identified in the preceding sections, we introduce Process-oriented Explainability (PoE), a transparency-by-design extension to existing practices. Rather than replacing traditional code diff-based validation, PoE adds a complementary layer that aims to make the agent’s global, process-level behaviour accessible to human supervisors. While classical diffs remain effective for experienced developers to inspect localized changes, agentic workflows increasingly involve planning steps, multi-file operations, and tool-mediated actions that fall outside the scope of line-by-line review. By foregrounding the process through which agents arrive at their modifications, this additional transparency layer supports coder-agent-reviewer interactions and provides meaningful epistemic benefits, especially for laypeople who rely on ‘vibe coding’ and lack the expertise to assess systemic impact. As software systems transition from static code toward autonomous agentic workflows, the primary diagnostic artifact necessarily shifts from traditional *stack traces*, which capture execution-level telemetry, to *reasoning traces*, which provide intent-level telemetry. The PoE framework leverages this shift by augmenting existing review practices with a global overview of agentic behaviour. For professional developers and reviewers, this provides a medium epistemic gain through improved oversight of agentic decision patterns; for lay users, it enables visibility into structural effects that would otherwise remain hidden behind functional correctness. Our framework is structured around three conceptual cornerstones.

I. Conformity structure. First, PoE requires **architectural seeding** to ground agentic behaviour in system-level constraints. Here, the human supervisor provides a machine-readable specification of global design patterns, invariants, or security requirements. Consistent with the perspective advanced by Singh et al. [17], this layer supports stakeholders (particularly software architects) who require explanations at a strategic rather than purely functional level. Rather than displacing code diffs, architectural seeds enrich them, as the agent must relate its proposed actions to these constraints, enabling reviewers to interpret local changes within a global architectural rationale. This reduces entropy-induced drift by ensuring that agentic plans remain aligned with design intent.

II. Reasoning monitoring. We extend transparency from code outputs to the **reasoning processes** that generate them. An instrumented environment captures the agent’s tool calls, intermediate planning steps, and contextual cues that motivate each action. This monitoring layer within PoE complements traditional review, where developers retain line-by-line inspection but also gain access to a global perspective on the agent’s decision trajectory [8]. For lay users, this layer exposes otherwise opaque behavioural patterns, reducing the risk that rapid prototyping or surface-level functionality masks accumulating architectural divergence.

III. Human interpretability layer. To render high-dimensional reasoning logs accessible, we summarize monitoring traces within PoE into a **causal reasoning graph** that depicts agentic behaviour as a sequence of reasoning nodes and

tool-mediated edges. By parsing the agent’s intermediate reasoning steps (including CoT-style segments in reasoning-centric models) into a directed acyclic graph (DAG), we can represent the agentic process as a series of connected nodes where each node signifies a possible agentic execution and each edge denotes a corresponding environmental action [7]. Reviewers can examine this graph alongside code diffs to identify deviations from architectural seeds or unexpected execution paths. By shifting cognitive effort from exhaustive local inspection to selective process-level oversight, this layer enhances a reviewer’s capacity to detect entropy-producing behaviours without replacing established practices. We add an *understanding requirement*, whereby the reviewer should be able to independently reconstruct the principal steps in the causal graph and recognize when an action falls outside permitted boundaries, with failure to do so signaling growing cognitive debt that warrants intervention [15].

Illustrative scenario. To ground the three pillars in a concrete setting, consider a web application whose architectural specification mandates that all database access be routed through a dedicated data-access layer. A user prompts an agentic coding tool to “add a caching layer to speed up the product catalog”. The agent, operating within a constrained context window, identifies the relevant API endpoint and introduces an in-memory cache. To populate the cache efficiently, it embeds a direct database query inside the API controller, bypassing the prescribed data-access layer entirely. The resulting code compiles, passes existing unit tests, and demonstrably improves response times, representing a functional success that would satisfy a vibe-coding user and appear benign in a standard code diff, which shows only the addition of a new import and a query call. Under PoE, this divergence surfaces at each pillar. The conformity structure (Pillar I) includes an architectural seed specifying that database interactions must pass through the data-access module; the agent’s proposed modification can therefore be evaluated against this constraint before it is committed. Reasoning monitoring (Pillar II) captures the agent’s intermediate plan, including the rationale “query the database directly from the controller for lower latency”, exposing the deliberate intent to circumvent the prescribed layer. This rationale, invisible in the final diff, becomes an explicit and auditable record of the agent’s decision trajectory. The causal reasoning graph (Pillar III) renders this sequence as a node whose stated intent (“direct DB access in controller”) conflicts with the architectural seed, flagging the deviation as a candidate for reviewer attention. A traditional diff-based review would present a syntactically reasonable, test-passing change; the PoE layer reveals that the agent’s reasoning trajectory departed from the system’s architectural invariant, precisely the kind of entropy-producing behaviour that, as discussed in Section 3, compounds silently across iterations and materializes as agentic technical debt.

Technical Implementation: The Reasoning-Action Loop. To operationalize PoE, we implement a reasoning–action loop within an instrumented execution environment. This environment acts as a transparent proxy around all tool-mediated interactions, such as file system operations, shell commands, or compiler invocations, allowing each step of agentic activity to be captured in relation to the agent’s intermediate reasoning. Each reasoning segment, tool call, and resulting environmental observation is recorded as a *reasoning trace node*. By linking these nodes into a DAG, the system provides a process-level representation of the agent’s behaviour that complements traditional code diffs. A ‘deviation detector’ compares each node’s stated intent against PoE’s architectural seeds defined in Pillar I, enabling early identification of entropy-inducing divergences in the agent’s plan [8]. To account for the human side of oversight, the same instrumentation should support a lightweight *cognitive debt index* that estimates how consistently reviewers engage with, and can reconstruct, the agent’s causal chain, helping to signal when human understanding begins to drift [15]. This implementation supplements existing review workflows by providing a global view of how autonomous actions unfold.

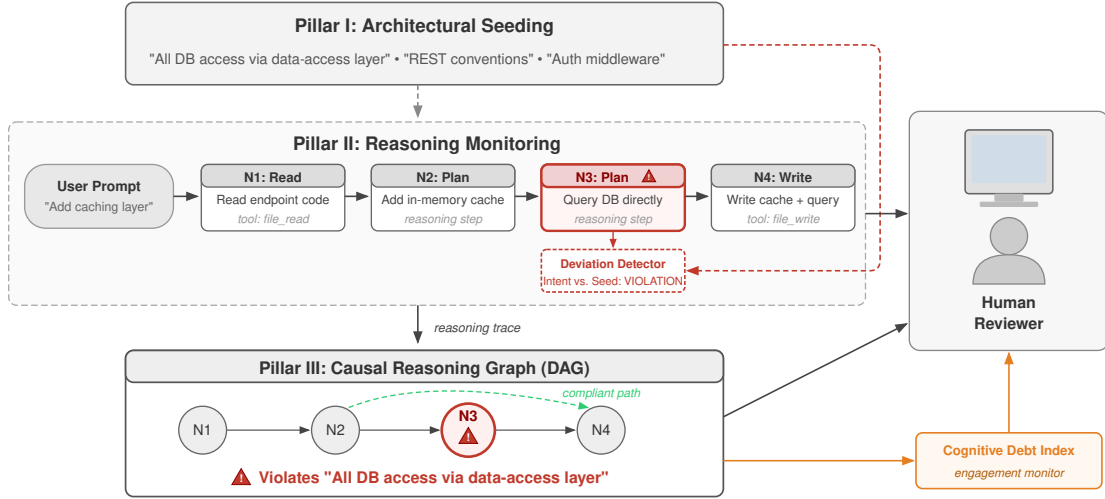


Fig. 1. The Process-oriented Explainability (PoE) framework applied to a data-access caching scenario. The figure illustrates the three pillars: Architectural Seeding, Reasoning Monitoring, and the Causal Reasoning Graph. It highlights how a deviation is detected when a reasoning step directly queries the database, violating the architectural rule. A cognitive debt index monitors reviewer engagement with the causal chain, signalling when human comprehension risks falling below the level required for substantive oversight.

5 Conclusion and Future Work

The HCXAI community must expand beyond localized transparency to address the broader challenge of agentic accountability. This work shows that without effective mitigation, the productivity gains of autonomous agents can be outweighed by the long-term costs of agentic entropy. By shifting attention from execution-level stack traces to intent-level reasoning traces, we offer a way to preserve architectural integrity as software systems evolve. Critically, the risks we identify are not limited to the codebase itself. As warned by Kosmyna et al. [12], Lin et al. [15], sustained AI delegation erodes the user’s ability to explain, verify, or intervene. In the context of agentic software development, this gap manifests as cognitive debt. Our PoE framework directly targets this dynamic. By making the agent’s causal reasoning chain legible and auditable, it supports what Lin et al. [15] define as the ‘Cognitive Integrity Threshold’ (CIT), i.e., the minimum level of comprehension required for oversight to remain substantive rather than performative. The cognitive debt index proposed in our implementation operationalizes this threshold in practice, flagging when reviewer engagement with the causal graph drops below a level sufficient to detect architectural drift. Future research should therefore treat cognitive drift as a first-class metric alongside code quality indicators, developing empirical benchmarks for CIT in agentic development contexts and studying how process-level interfaces can actively counteract comprehension decay over time.

Future research should prioritize the development of *Standardized Agentic Traces*, representing a universal, schema-agnostic log format. Such a standard would facilitate interoperability across agents and human-centric audit tools, ensuring that reasoning remains transparent across diverse development environments. In addition, empirical studies are needed to quantify the reduction in cognitive load provided by process-level ‘cognitive debugging’ interfaces compared to traditional code review practices [5]. Ultimately, the goal of agentic XAI is to shift the human-agent relationship from one of blind trust in velocity to one of shared, verifiable understanding of architectural intent, an understanding that must be actively maintained in both the codebase and the minds of the people who oversee it.

Acknowledgments

This work was partly conducted within the framework of the EUonAIR Centre of Excellence in Responsible AI and Education. The authors have been supported by a grant from Movetia, which is funded by the Swiss Confederation.

References

- [1] Ofra Amir. 2021. Conveying Agent Behavior to People. <https://hcxai.jimdosite.com>
- [2] Shraddha Barke, Michael B. James, and Nadia Polikarpova. 2023. Grounded Copilot: How Programmers Interact with Code-Generating Models. *Proc. ACM Program. Lang.* 7, OOPSLA1, Article 78 (4 2023), 27 pages. doi:10.1145/3586030
- [3] DORA Team at Google Cloud. 2024. *2024 Accelerate State of DevOps*. Annual Research Report. Google Cloud, Sunnyvale, CA, USA. <https://dora.dev/research/2024/dora-report/2024-dora-accelerate-state-of-devops-report.pdf>
- [4] Xueying Du, Mingwei Liu, Kaixin Wang, Hanlin Wang, Junwei Liu, Yixuan Chen, Jiayi Feng, Chaofeng Sha, Xin Peng, and Yiling Lou. 2024. Evaluating Large Language Models in Class-Level Code Generation. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering (Lisbon, Portugal) (ICSE '24)*. Association for Computing Machinery, New York, NY, USA, Article 81, 13 pages. doi:10.1145/3597503.3639219
- [5] Sarah Fakhoury, Aaditya Naik, Georgios Sakkas, Saikat Chakraborty, Madan Musuvathi, and Shuvendu Lahiri. 2024. Exploring the Effectiveness of LLM based Test-driven Interactive Code Generation: User Study and Empirical Evaluation. In *Proceedings of the 2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings (Lisbon, Portugal) (ICSE-Companion '24)*. Association for Computing Machinery, New York, NY, USA, 390–391. doi:10.1145/3639478.3643525
- [6] GitClear Research. 2025. AI Copilot Code Quality: 2025 Look Back at 12 Months of Data. https://www.gitclear.com/ai_assistant_code_quality_2025_research
- [7] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. 2025. DeepSeek-R1 Incentivizes Reasoning in LLMs Through Reinforcement Learning. *Nature* 645, 8081 (2025), 633–638. doi:10.1038/s41586-025-09422-z
- [8] Ahmed E. Hassan, Hao Li, Dayi Lin, Bram Adams, Tse-Hsun Chen, Yutaro Kashiwa, and Dong Qiu. 2025. Agentic Software Engineering: Foundational Pillars and a Research Roadmap. arXiv:2509.06216 [cs.SE]
- [9] Saffron Huang, Bryan Seethor, Esin Durmus, Kunal Handa, Miles McCain, Michael Stern, and Deep Ganguli. 2025. How AI Is Transforming Work at Anthropic. <https://anthropic.com/research/how-ai-is-transforming-work-at-anthropic>
- [10] Samia Kabir, David N. Udo-Imeh, Bonan Kou, and Tianyi Zhang. 2024. Is Stack Overflow Obsolete? An Empirical Study of the Characteristics of ChatGPT Answers to Stack Overflow Questions. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems (Honolulu, HI, USA) (CHI '24)*. Association for Computing Machinery, New York, NY, USA, Article 935, 17 pages. doi:10.1145/3613904.3642596
- [11] Shaun Khoo, Jessica Foo, and Roy Ka-Wei Lee. 2025. With Great Capabilities Come Great Responsibilities: Introducing the Agentic Risk & Capability Framework for Governing Agentic AI Systems. arXiv:2512.22211 [cs.AI]
- [12] Nataliya Kosmyna, Eugene Hauptmann, Ye Tong Yuan, Jessica Situ, Xian-Hao Liao, Ashly Vivian Beresnitzky, Iris Braunstein, and Pattie Maes. 2025. Your Brain on ChatGPT: Accumulation of Cognitive Debt when Using an AI Assistant for Essay Writing Task. arXiv:2506.08872 [cs.AI]
- [13] Meir M. Lehman. 1980. Programs, Life Cycles, and Laws of Software Evolution. *Proc. IEEE* 68, 9 (1980), 1060–1076. doi:10.1109/PROC.1980.11805
- [14] Hao Li, Haoxiang Zhang, and Ahmed E. Hassan. 2025. The Rise of AI Teammates in Software Engineering (SE) 3.0: How Autonomous Coding Agents Are Reshaping Software Engineering. arXiv:2507.15003 [cs.SE]
- [15] Fangzhou Lin, Qianwen Ge, Lingyu Xu, Peiran Li, Xiangbo Gao, Shuo Xing, Kazunori Yamada, Ziming Zhang, Haichong Zhang, and Zhengzhong Tu. 2026. Position: Human-Centric AI Requires a Minimum Viable Level of Human Understanding. arXiv:2602.00854 [cs.AI]
- [16] Shaina Raza, Ranjan Sapkota, Manoj Karkee, and Christos Emmanouilidis. 2025. TRiSM for Agentic AI: A Review of Trust, Risk, and Security Management in LLM-based Agentic Multi-Agent Systems. arXiv:2506.04133 [cs.AI]
- [17] Ronal Singh, Upol Ehsan, Marc Cheong, Mark O. Riedl, and Tim Miller. 2021. LEx: A Framework for Operationalising Layers of AI Explanations. <https://hcxai.jimdosite.com>
- [18] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2022. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. In *Proceedings of the 36th International Conference on Neural Information Processing Systems (New Orleans, LA, USA) (NIPS '22)*. Curran Associates Inc., Red Hook, NY, USA, Article 1800, 14 pages. doi:10.5555/3600270.3602070