

Revealing the Dark Matter: Connecting Tacit and System Knowledge in Human-AI Collaborations

Katherine R. Dearstyne
University of Notre Dame
Notre Dame, IN, USA
kdearsty@nd.edu

Carmen Badea, Christian Bird, Robert DeLine
Microsoft
Redmond, WA, USA
cabadea,cbird,rdeline@microsoft.com

Abstract

Software processes rely on both structured system knowledge such as code, version histories, and logs, and tacit knowledge including human rationale, practices, and decisions. We argue that effective human-AI collaboration requires shared and evolving knowledge spaces that integrate these knowledge sources and make their connections explicit. Using differential testing as a motivating case, we describe our initial prototype and the challenges it revealed, and we outline a broader vision for dynamic knowledge networks that support more effective collaboration between humans and AI.

1 Introduction

For decades software engineering research has primarily focused on “*system knowledge*”: structured data such as source code, bug databases, and version control histories. The software engineering community have shown what can be achieved when these artifacts are systematically analyzed [3, 10, 12]. While valuable, such efforts overlook the “*dark matter*” of software engineering: the tacit knowledge, informal rationale, and everyday decisions conveyed through fleeting exchanges such as hallway conversations or impromptu design discussions. Even when some of this knowledge was captured, it often remained buried in unstructured forms such as wiki pages, notes, or informal chats, with no reliable way to connect it to other artifacts and contextualize it within the broader system [2, 18].

Two recent shifts create new opportunities to capture and harness this previously buried knowledge. The first is the widespread move to remote and hybrid work which has led to an unprecedented amount of recorded communication. The second is the rise of LLMs, which can rapidly ingest, summarize, and reason over vast amounts of unstructured text, including emails, team docs, online chats, and meeting transcripts [1, 21]. This allows them to trace connections across diverse software artifacts at speeds impossible for human engineers [14, 26]. This convergence of newly available unstructured data with AI’s capacity for rapid reasoning creates a path to integrate tacit and system knowledge within shared knowledge spaces, enriching both human and agent understanding of software systems [20]. Ideally, this space should evolve alongside the system, continually capturing insights from ongoing use and revealing additional implicit knowledge over time.

To begin exploring this vision, we focus on differential testing in software release engineering, which served as our motivating case [8, 11, 23] as it exemplifies tasks that require rapidly integrating and reasoning over vast and varied sources of knowledge. We first describe our initial prototype that integrates both tacit and system knowledge, reflect on the challenges we encountered, and then outline how these challenges point to a broader research agenda. Building on these lessons, we conclude with a vision for dynamic knowledge networks and process analysis techniques that surface tacit and system knowledge, connect them with established artifacts, and enable more effective and adaptable human-agent collaboration across software engineering tasks.

2 Motivating Example: Differential Testing

Differential testing [8, 11, 23] is a technique that compares the behavior of production and test systems to catch regression failures. A single build can generate hundreds of behavioral differences (diffs), creating a large volume of work in which each diff must be analyzed to determine whether it reflects a true regression or instead arises from a benign source, such as a newly introduced feature or noise from non-determinism. The scale and repetitive nature of this task make it a strong candidate for automation, yet interviews with differential testing engineers revealed that automation alone is insufficient [11]. As one engineer explained, “I need to know what data led to the prediction...without that, I’d still have to do all the investigation myself” [11]. An effective solution therefore requires human-agent partnership where agents first sift through vast amounts of data, including both system knowledge and tacit knowledge, to identify possible sources of diff causes, providing crucial connections and context for humans to quickly review and confirm the analysis.

2.1 Initial Solution

Our initial solution explored an agentic framework that provided the LLM with tools that mirror the resources typically available to engineers. Through interviews with engineers, we identified 9 diverse sources of knowledge that were relied upon during diff investigations, ranging from team knowledge to system knowledge, as summarized below:

Team Knowledge: Wiki pages, team messages, meeting transcripts.

Version Control: Commit/PR messages, code diffs.

Codebase: Source code, error metadata.

Historical: Labeled diffs, notes from prior builds.

To leverage this information, we gave the agent direct access to some data sources through database and version control repository retrieval. For other data sources, the agent could make high-level queries using GraphRAG[7]. Among these sources, team knowledge proved particularly critical, since it allowed the transfer of human expertise *asynchronously* to the agent. This category included resources such as Wiki pages and team messages in the initial prototype, with meeting transcripts considered as part of our broader research vision (see Section 3). By providing this context upfront, the agent was able to reduce the degree of human intervention required at runtime because it already had access to much of the tacit knowledge that engineers typically convey through ongoing communication and informal interactions.

At each step of the diff investigation, the agent could make tool calls to gather new information, either allowing it to reach a conclusion or helping it decide what additional calls were needed to explore the problem further. After each step the agent was required

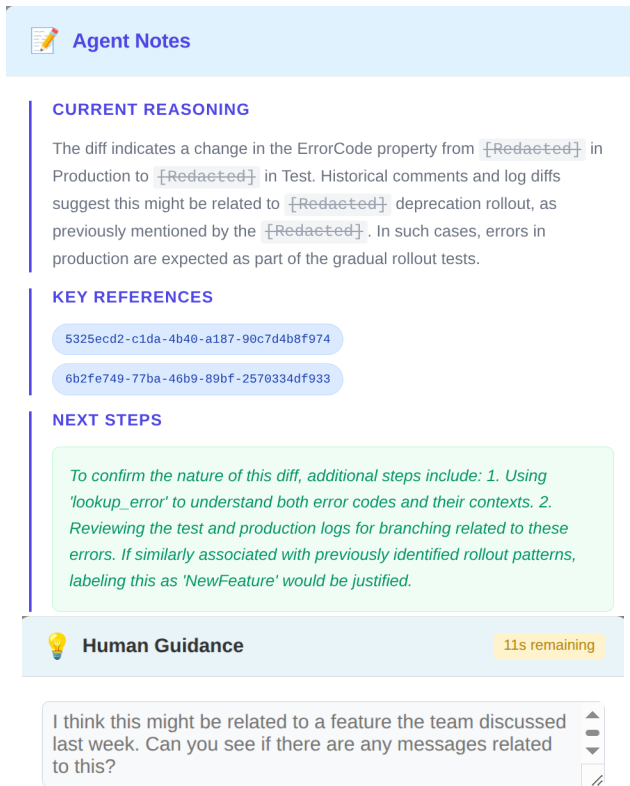


Figure 1: Example of a reasoning step provided by the diff investigator agent’s notes (top) and the human input provided by the user in response (bottom).

to provide notes documenting what new information it had uncovered, how that information updated its beliefs about the root cause, and what next steps it planned to take. To ensure traceability to specific software artifacts, we labelled each piece of information with a UUID when providing it to the agent and we required the agent to include the relevant UUIDs when providing its reasoning. These notes were presented to the user in real time, with each reference presented as a clickable link to the details of the associated artifact (see Figure 1).

Once the agent had recorded its notes, we provided an optional space for the user to give input to the agent. This allowed them to redirect the agent when it began heading down unproductive paths, or to strengthen the agent’s investigation with human expertise. This *synchronous* form of communication complemented the *asynchronous* team knowledge by pairing real-time human guidance with knowledge distilled from past human communication.

2.2 Challenges

While the initial prototype was a step toward uniting agent and human knowledge, we encountered several challenges that prompted us to consider a broader design vision for the solution.

Challenge 1: Accuracy/Transparency Trade-offs

Although the agentic approach provided contextual information and reasoning, we found that it sacrificed some of the high accuracy achieved by a GPT-4 model fine-tuned to classify diffs as noise, features, or regressions [23]. This likely occurs because the fine-tuned model, trained on a large volume of data, can identify

patterns that are not apparent when examining only a few examples, but these patterns are difficult to use because we cannot interpret what the model has learned. Although we did provide the agent with access to past examples, supplying enough examples to fully recognize the same patterns as the fine-tuned model would exceed the context window limit. Instead, the agent often over-relied on spurious patterns found in the provided examples.

Challenge 2: Knowledge Retention and Reuse

Engineers improve at tasks such as differential testing through practice as they recognize common patterns that emerge over time. However, in the current implementation, the agent had to perform the process from scratch without leveraging prior work. One approach to utilize past attempts is through fine-tuning an LLM [23], but there are also use cases where information from one investigation might be useful at a human-interpretable level for future tasks. For example, when a similar diff appears, it might be sufficient to reference previous reasoning rather than re-running the agent. Alternatively, the information might be relevant outside the current task entirely, such as when code files identified during a "new feature" diff investigation prove useful for change impact analysis of that same feature. A mechanism is needed to efficiently store and leverage task knowledge for downstream applications.

Challenge 3: Managing Information Scale and Context

The amount of information relevant to the task (e.g., code files, logs, commits) quickly blows out the context window. One way to reduce context is through summarization [24], while another is to offload some of the context to an agent that can specialize in it [28]. However, both approaches risk information loss where important details may not reach the primary agent, and both involve additional computational steps that slow down the overall process. This created a constant challenge of balancing information completeness with efficiency.

Challenge 4: Routine Steps vs. Flexible Problem-Solving

Granting the agent freedom to make tool calls based on current information allowed it to build on previous findings and focus sequentially on specific pieces of information. However, this flexibility came at significant cost: tool calls could not be parallelized, slowing the overall process, and the agent sometimes made unnecessary tool calls. This contrasts with how human experts approach investigations. Engineers often have predefined steps they perform in certain contexts (e.g., start with comparing error codes, or examine specific log sections). This recipe-based approach works well as a starting place when ambiguity about the cause is high. However, as new information emerges, experts might diverge to new plans that are no longer formulaic, requiring flexible adaptation based on their findings. To accelerate the process, we needed to mirror this approach by identifying routine steps that could be performed through predefined workflows, versus those that would benefit from full autonomy.

Challenge 5: Tacit Knowledge Transfer

We found that some of the human experts’ knowledge about the decision-making process was tacit, making it difficult to convey to the agent. This unspoken expertise includes intuitions about which information sources are most reliable, how to interpret ambiguous signals, and when certain patterns indicate specific types of problems. We needed a way to transfer this implicit

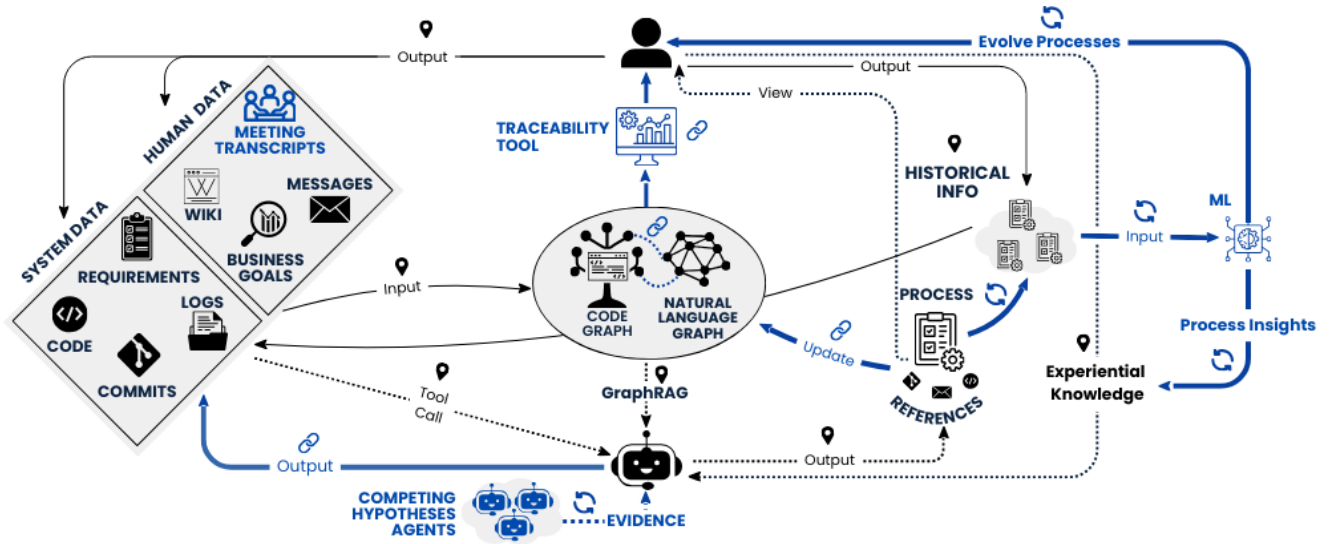


Figure 2: Research vision of an evolving knowledge space shared by humans and agents. Black nodes with thin lines (●) show the initial prototype; blue nodes with thick lines represent the next phase. Dotted lines indicate synchronous communication, solid lines asynchronous. The 🔗 marks “Dynamic Relationships,” and the 🔄 marks “Process Analysis & Evolution.”

expert knowledge to the agent so that it could perform more efficient, successful investigations. Team messages and wiki pages helped alleviate part of this challenge by capturing some of the expertise, but other forms of communication (e.g., meetings) also conferred important knowledge, and some expertise was never articulated at all. Interestingly, this represents almost the reverse of Challenge 1, where patterns learned by the fine-tuned model were unable to be communicated.

Challenge 6: Confirmation Bias and Hypothesis Anchoring

We observed that LLM agents exhibited strong confirmation bias. Once they found any early evidence supporting a particular label, they would rarely stray from that interpretation despite contradictory evidence encountered later. This anchoring effect is particularly problematic in differential testing where incomplete or misleading initial information can misdirect entire investigations. Unlike humans who can reconsider assumptions when prompted, LLMs remain committed to their early conclusions.

3 Research Vision

Underlying each of these challenges are fundamental barriers to knowledge transfer, whether between recipients (humans vs. AI), contexts (different tasks), or timeframes (past to present). Our vision (Figure 2) aims to overcome these challenges by surfacing this “dark matter” knowledge through a unified knowledge network that connects diverse software artifacts. Such a network would evolve alongside the system and provide deeper insight into software processes themselves.

Starting Point: Prototype (●) Our initial prototype established the foundation for this vision, incorporating not only system-generated artifacts but also human-generated communications such as message exchanges between team members. These *asynchronous* communications provide insight into knowledge that may not be explicitly captured in traditional system artifacts. The prototype also supported *synchronous* human input collected at runtime, allowing


experiential knowledge to be conveyed throughout the investigation. Additionally, past knowledge was transferred by supplying the agent with examples of similar, previously labeled data when requested. Conversely, communication from the agent to humans occurred primarily through note-taking, where the agent documented its reasoning and explicitly referenced project data for review. These initial communication mechanisms establish the foundation on which the broader vision can be built.

Building: Dynamic Relationships (🔗) This starting point brought together a variety of software artifacts and human communications, with some initial connections based on semantic similarity and AST information. However, this still falls short of conveying a complete picture of how different artifact types relate to one another. Semantic similarity is inherently limited in the types of relationships it can identify [16, 25], while AST information captures structural details without revealing how artifacts connect within the broader system. Building a more comprehensive network would create a knowledge base that enables deeper system understanding, maintains connections that persist across different contexts (Challenge 2), and minimizes the need to search through vast information spaces (Challenge 3). Importantly, this expanded knowledge network could begin to address Challenge 5 by integrating a wider range of artifacts, incorporating team communications such as meeting transcripts or past human interventions alongside more traditional sources.

A complementary path for discovering meaningful relationships is through the natural interactions that occur when completing a task [9]. As agents perform tasks and identify relevant data sources, these interactions can be recorded, forming links between diverse artifacts and contexts. Such links can evolve over time as agents revisit and refine earlier relationships, while connections that prove unhelpful can be weakened or removed. This enables the construction of knowledge networks that are continuously updated and improved through ongoing use.

As the network evolves, humans can engage with the links

through traceability tools [17, 22], enabling more effective knowledge transfer and oversight. As AI agents contribute to increasingly large portions of the system, these dynamic links will become even more vital for transferring knowledge back to human collaborators and ensuring accountability of the agent’s outcomes. Additionally, these links can create benefits beyond AI-human interactions, providing valuable insights for other tasks, such as developer onboarding or safety assessments [5].

Building: Process Analysis & Evolution  The original version also incorporated knowledge in the form of past examples, which can illuminate hidden patterns underlying expert intuitions [19]. Yet, this data was underutilized in the initial implementation. Examples were either provided to the agent in small subsets where broader generalizations were impossible, or used as training data for black-box models that could not reveal the learned patterns (Challenge 1). Another overlooked source of knowledge lies in agent traces, which contain a wealth of information, including reasoning processes, steps taken, tools invoked, data sources consulted, and points of human intervention. When analyzed effectively, this information can reveal which aspects of the software process succeed or fail, enabling refinements that better balance efficiency with exploration (Challenge 4). By systematically surfacing patterns in these process artifacts, expert knowledge can be made explicit for both humans and agents (Challenge 5).

To this end, statistical analysis has already demonstrated promise in uncovering meaningful patterns in reasoning trajectories linked to success or failure [4], as well as causal relationships between agent behaviors [13]. Building on this, additional approaches can quantify the contribution of specific data sources to information gain, enabling more efficient input prioritization while reducing context demands [15]. They can also track the frequency of steps or tool calls to identify candidates for predefined workflows, which has the potential to improve performance and efficiency [27]. Finally, they can capture recurring input from humans at runtime, allowing these inputs to be reused without requiring repeated human intervention.

To address hypothesis anchoring (Challenge 6), we propose using Analysis of Competing Hypotheses (ACH) [6] through a multi-agent architecture. When a software process begins, the system launches multiple agents, each advocating for a different hypothesis; in differential testing, for example, one agent would represent each possible classification for a diff (regression, new feature, and noise). Each agent independently gathers evidence to support its assigned hypothesis, ensuring all explanations receive thorough consideration. The agents then present their evidence to either human engineers or an LLM judge for final evaluation. This approach reduces bias by separating hypothesis generation from evidence gathering and improves transparency by allowing side-by-side comparison of competing explanations. The system also generates valuable training data by tracking which hypotheses prove correct over time, helping optimize future investigations.

Crucially, these insights extend beyond improving agent processes. Inspired by prior work showing that experts can learn from AI knowledge [20], we view this as a mechanism for refining human understanding and decision-making. By accumulating and analyzing process patterns, engineers gain a deeper understanding of how tasks are executed, enabling them to refine, optimize, and ultimately improve their own workflows. In this way, process knowledge flows both ways, helping humans and agents alike increase their chances

of success.

4 Future Plans

Our immediate roadmap focuses on implementing and evaluating the key components of our unified knowledge network vision, as follows:

1. Knowledge Network Storage and Management

- *Implement storage mechanisms* (e.g., graph databases) to capture and maintain the artifact relationships identified through agent interaction and human feedback.
- *Support dynamic adjustment of relationship strengths* based on task outcomes, and enable pruning of outdated connections flagged by agents during task execution.
- *Introduce relationship typing* to distinguish between structural dependencies, semantic similarities, and task-related associations.

Key Challenges: Scalability of data storage and retrieval; accuracy of agent’s evaluation of what information was important to the task; maintaining network quality as relationships multiply.

2. Process Pattern Analysis

- *Compare statistical approaches for identifying patterns* in agent reasoning trajectories, tool usage, and decision points linked to task success or failure.
- *Create systems to track frequency and effectiveness* of investigative steps, identifying candidates for predefined workflows and comparing performance with full autonomy approaches.
- *Evaluate different methods for incorporating learned patterns* into process improvements, including reinforcement learning and prompt engineering techniques.
- *Implement a multi-agent architecture* and evaluate its effectiveness in reducing hypothesis anchoring compared to single-agent approaches.

Key Challenges: Ensuring patterns remain generalizable and interpretable across contexts; ensuring interpretability and usefulness of discovered patterns to human engineers.

3. Validation & Refinement

- *Apply the approach to different software processes* beyond differential testing, such as change impact analysis or developer task prioritization, to evaluate generalizability and cross-context effectiveness.
- *Conduct user studies* to examine perceived usefulness of the approach for human-AI interactions and identify opportunities to improve user experience.
- *Compare performance* of the agent-based approach with fine-tuned models and other baseline methods across multiple evaluation metrics.

Key Challenges: Designing metrics that capture both task performance and knowledge utility; adapting methods to varied software processes without losing effectiveness or overfitting to specific workflows.

5 Conclusion

Software processes rely on both structured system knowledge and tacit human knowledge, yet these sources often remain disconnected, limiting the transfer of insights across tasks and individuals. Our example in differential testing illustrated both the potential

and the challenges of unifying them. To address these barriers, we propose dynamic knowledge networks that link diverse artifacts and make process insights explicit. Such networks provide a foundation for more effective human-AI collaboration, where shared knowledge spaces evolve with software systems and strengthen both human understanding and agent performance.

References

- [1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altmenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774* (2023).
- [2] Jorge Aranda and Gina Venolia. 2009. The secret life of bugs: Going past the errors and omissions in software repositories. In *2009 IEEE 31st international conference on software engineering*. IEEE, 298–308.
- [3] Stefan Biffl, Wikan Danar Sunindyo, and Thomas Moser. 2010. Semantic Integration of Heterogeneous Data Sources for Monitoring Frequent-Release Software Projects. In *2010 International Conference on Complex, Intelligent and Software Intensive Systems*. 360–367. doi:10.1109/CISIS.2010.58
- [4] Islem Bouzenia and Michael Pradel. 2025. Understanding Software Engineering Agents: A Study of Thought-Action-Result Trajectories. *arXiv preprint arXiv:2506.18824* (2025).
- [5] Jane Cleland-Huang, Orlena CZ Gotel, Jane Huffman Hayes, Patrick Mäder, and Andrea Zisman. 2014. Software traceability: trends and future directions. In *Future of software engineering proceedings*. 55–69.
- [6] Mandeep K Dhami, Ian K Belton, and David R Mandel. 2019. The “analysis of competing hypotheses” in intelligence analysis. *Applied Cognitive Psychology* 33, 6 (2019), 1080–1090.
- [7] Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, Dasha Metropolitan, Robert Osazuwa Ness, and Jonathan Larson. 2024. From local to global: A graph rag approach to query-focused summarization. *arXiv preprint arXiv:2404.16130* (2024).
- [8] Muhammad Ali Gulzar, Yongkang Zhu, and Xiaofeng Han. 2019. Perception and Practices of Differential Testing. (2019), 71–80. doi:10.1109/ICSE-SEIP.2019.00016
- [9] Paul Hübner and Barbara Paech. 2017. Using Interaction Data for Continuous Creation of Trace Links Between Source Code and Requirements in Issue Tracking Systems. *Lecture Notes in Computer Science* 10153, 291–307. doi:10.1007/978-3-319-54045-0_21
- [10] Huzefa Kagdi, Michael L Collard, and Jonathan I Maletic. 2007. A survey and taxonomy of approaches for mining software repositories in the context of software evolution. *Journal of software maintenance and evolution: Research and practice* 19, 2 (2007), 77–131.
- [11] Arun Krishna Vajjala, Ajay Krishna Vajjala, Carmen Badea, Christian Bird, Jade D’Souza, Robert DeLine, Mikhail O Demyanyuk, Jason Entenmann, Nicole Forsgren, Aliaksandr Hramadski, Haris Mohammad, Sandeepan Sanyal, Oleg Surmachev, and Thomas Zimmermann. 2025. Using Large Language Models to Support the Workflow of Differential Testing. In *Proceedings of the 33rd ACM International Conference on the Foundations of Software Engineering* (Clarion Hotel Trondheim, Trondheim, Norway) (*FSE Companion ’25*). Association for Computing Machinery, New York, NY, USA, 355–365. doi:10.1145/3696630.3728559
- [12] Meir M. Lehman and Juan F. Ramil. 2002. Software Evolution and Software Evolution Processes. *Annals of Software Engineering* 14, 1 (Dec. 2002), 275–309. doi:10.1023/A:1020557525901
- [13] Jiaying Lu, Bo Pan, Jieyi Chen, Yingchaojie Feng, Jingyuan Hu, Yuchen Peng, and Wei Chen. 2024. Agentlens: Visual analysis for agent behaviors in llm-based autonomous systems. *IEEE Transactions on Visualization and Computer Graphics* (2024).
- [14] Tula Masterman, Sandi Besen, Mason Sawtell, and Alex Chao. 2024. The landscape of emerging ai agent architectures for reasoning, planning, and tool calling: A survey. *arXiv preprint arXiv:2404.11584* (2024).
- [15] Varatheepan Paramanayakam, Andreas Karatzas, Iraklis Anagnostopoulos, and Dimitrios Stamoulis. 2025. Less is More: Optimizing Function Calling for LLM Execution on Edge Devices. In *2025 Design, Automation Test in Europe Conference (DATE)*. 1–7. doi:10.23919/DATE64628.2025.10992798
- [16] Alberto D. Rodriguez, Katherine R. Dearstyne, and Jane Cleland-Huang. 2023. Understanding the Challenges of Deploying Live-Traceability Solutions. *arXiv:2306.10972* [cs.SE] <https://arxiv.org/abs/2306.10972>
- [17] Alberto D. Rodriguez, Timothy Newman, Katherine R. Dearstyne, and Jane Cleland-Huang. 2023. SAFA: A Tool for Supporting Safety Analysis in Evolving Software Systems. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering* (Rochester, MI, USA) (*ASE ’22*). Association for Computing Machinery, New York, NY, USA, Article 165, 4 pages. doi:10.1145/3551349.3559535
- [18] Sharon Ryan and Rory V. O’Connor. 2013. Acquiring and sharing tacit knowledge in software development teams: An empirical study. *Information and Software Technology* 55, 9 (2013), 1614–1624. doi:10.1016/j.infsof.2013.02.013
- [19] Eduardo Salas, Michael A. Rosen, and Deborah DiazGranados. 2010. Expertise-Based Intuition and Decision Making in Organizations. *Journal of Management* 36, 4 (2010), 941–973. [arXiv:https://doi.org/10.1177/0149206309350084](https://doi.org/10.1177/0149206309350084) doi:10.1177/0149206309350084
- [20] Lisa Schut, Nenad Tomasev, Tom McGrath, Demis Hassabis, Ulrich Paquet, and Been Kim. 2023. Bridging the Human-AI Knowledge Gap: Concept Discovery and Transfer in AlphaZero. *arXiv:2310.16410* [cs.AI] <https://arxiv.org/abs/2310.16410>
- [21] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971* (2023).
- [22] Hanny Tufail, Muhammad Faisal Masood, Babar Zeb, Farooque Azam, and Muhammad Waseem Anwar. 2017. A systematic review of requirement traceability techniques and tools. In *2017 2nd International Conference on System Reliability and Safety (ICSRS)*. 450–454. doi:10.1109/ICSRS.2017.8272863
- [23] Ajay Krishna Vajjala, Arun Krishna Vajjala, Carmen Badea, Christian Bird, Jade D’Souza, Robert DeLine, Mikhail Demyanyuk, Jason Entenmann, Nicole Forsgren, Aliaksandr Hramadski, Haris Mohammad, Sandeepan Sanyal, Oleg Surmachev, and Thomas Zimmermann. 2025. Enhancing Differential Testing: LLM-Powered Automation in Release Engineering. In *2025 IEEE/ACM 47th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. 607–617. doi:10.1109/ICSE-SEIP66354.2025.00059
- [24] Cangqing Wang, Yutian Yang, Ruisi Li, Dan Sun, Ruicong Cai, Yuzhu Zhang, and Chengqian Fu. 2024. Adapting LLMs for Efficient Context Processing through Soft Prompt Compression. In *Proceedings of the International Conference on Modeling, Natural Language Processing and Machine Learning* (Xi’an, China) (*CMNM ’24*). Association for Computing Machinery, New York, NY, USA, 91–97. doi:10.1145/3677779.3677794
- [25] Orion Weller, Michael Boratko, Iftekhar Naim, and Jinhyuk Lee. 2025. On the Theoretical Limitations of Embedding-Based Retrieval. *arXiv:2508.21038* [cs.IR] <https://arxiv.org/abs/2508.21038>
- [26] Shangyu Wu, Ying Xiong, Yufei Cui, Haolun Wu, Can Chen, Ye Yuan, Lianming Huang, Xue Liu, Tei-Wei Kuo, Nan Guan, and Chun Jason Xue. 2025. Retrieval-Augmented Generation for Natural Language Processing: A Survey. *arXiv:2407.13193* [cs.CL] <https://arxiv.org/abs/2407.13193>
- [27] Chunqiu Steven Xia, Yinlin Deng, Soren Dunn, and Lingming Zhang. 2024. Agentless: Demystifying LLM-based Software Engineering Agents. *arXiv:2407.01489* [cs.SE] <https://arxiv.org/abs/2407.01489>
- [28] Yusen Zhang, Ruoxi Sun, Yanfei Chen, Tomas Pfister, Rui Zhang, and Sercan Arik. 2024. Chain of agents: Large language models collaborating on long-context tasks. *Advances in Neural Information Processing Systems* 37 (2024), 132208–132237.