Towards an Architecture Theory for Trustworthy Operational Knowledge Construction

Erik Nedeljković, Tihana Galinac Grbac

September 10, 2026

Abstract

Artificial Intelligence (AI) systems increasingly participate in operational decision-making across all domains, from various public and commercial services to critical infrastructures. Although the related technologies bring significant benefits in constructing operational interpretations from heterogeneous observations, it remains a serious concern the trustworthiness of operational knowledge constructed to support decision-making in the real operational environments, especially those that operate on real hardware. The increasing use of AI represents a new architectural challenge that shifts the architectural focus from decision making to knowledge construction and operational reasoning. Thus, we define operational knowledge construction as an emergent system property. We propose a failure-driven reasoning process to derive architectural responsibilities supporting trustworthy operational knowledge construction. The proposed perspective is illustrated using a representative 5G installation scenario, in which field engineers continuously integrate heterogeneous observations, construct operational understanding, and make decisions under uncertainty. Its purpose is to demonstrate on simple case the notion of operational knowledge construction as an emergent system property and the use of a proposed failure-driven reasoning process that shows how failures in operational knowledge construction can be translated into architectural responsibilities and thus contribute to system trustworthiness. We argue that the proposed perspective is intended to stimulate future research on software architectures supporting trustworthy human reliance on AI-generated operational knowledge and provides a conceptual foundation for responsible software architectures across critical infrastructure domains.

Keywords: software architecture, trustworthy, operational knowledge construction, emergent behaviour, failure-driven method, situational awareness

1 Introduction

Operational software systems are undergoing a fundamental transformation [1, 2]. Software traditionally supported operational activities by acquiring measurements, coordinating distributed components, and presenting information to human operators, while operational decisions remained the responsibility of human experts [2]. Contemporary software no longer merely presents operational information but increasingly participates in its interpretation. Modern AI-assisted systems retrieve documentation, analyse heterogeneous observations, generate explanations, propose corrective actions, and increasingly influence the operational decisions made by human operators [3]. As a consequence, software is progressively becoming an active participant in operational reasoning [1, 4].

This transition introduces a new architectural challenge [4]. Classical software architecture methods were developed assuming that software processes information whose semantics are either predefined [5] or explicitly modeled [6]. In AI-assisted systems, however, the software increasingly constructs operational knowledge by combining runtime observations, historical information, engineering documentation, contextual evidence, and probabilistic reasoning [3]. This changes the software architectural problem. This challenge can be illustrated using the commissioning of a 5G base station, where engineers combine heterogeneous observations before deciding whether the installation satisfies operational objectives. Although the example originates from telecommunications, similar workflows appear across AI-assisted critical infrastructures.

Existing research provides complementary foundations for addressing this challenge. Software architecture explains how software systems satisfy architectural concerns and quality attributes [7, 8]. Dependability and safety engineering investigate operational failures and hazards [9]. Human factors research explains how operators construct situational awareness [10], while trustworthy AI research investigates explainability, transparency, human oversight, and trust calibration [11–13]. Although these research directions contribute essential concepts, they provide limited guidance on how operational knowledge failures should be systematically transformed into architectural responsibilities supporting AI-assisted operational decision making.

This paper is intentionally conceptual. Rather than proposing or evaluating a complete software architecture, it formulates an emerging architectural problem introduced by AI-assisted operational decision-making and proposes an initial failure-driven reasoning framework for deriving architectural responsibilities supporting trustworthy operational knowledge construction. The contributions of this paper are therefore threefold. First, we formulate trustworthy operational knowledge construction as an emerging architectural concern arising from AI-assisted operational decision making in Sect.2. Second, we propose an initial failure-driven reasoning process for deriving architectural responsibilities supporting trustworthy operational knowledge construction in Sect. 3. Third, we demonstrate this reasoning process on a representative operational workflow and discuss its implications for future research on software architectures supporting AI-enabled critical infrastructures in Sect. 4. Finally, in Sect. 5 we

discuss findings and open discussions for future work. In Sect. 6 we conclude this paper.

2 Trustworthy Operational Knowledge Construction

In this paper, *AI-assisted operational systems* denote software-intensive systems in which AI-generated interpretations or recommendations influence human decisions affecting physical or organisational processes. Typical examples include telecommunications infrastructures, manufacturing plants, transportation systems, energy networks, and water distribution systems. Such environments increasingly operate as interconnected socio-technical systems integrating sensing, communication, computation, physical processes, and human operators [1]. Operational decisions in these systems are rarely based on isolated measurements. Human operators construct an understanding of the current situation by combining observations, engineering knowledge, historical information, procedures, and contextual evidence [10,14]. AI, and particularly generative AI, increasingly participates in this process by analysing heterogeneous information, retrieving relevant knowledge, and generating interpretations or recommendations [3, 4].

This process is closely related to Situation Awareness (SA), commonly characterised through perception of relevant elements, comprehension of their meaning, and projection of their future status [10, 15]. In software-intensive operational environments, these activities are increasingly mediated by software that acquires, integrates, and presents information used by human operators [16]. As runtime observations and operating conditions change, the knowledge representing the current situation must evolve accordingly [6]. SA is therefore relevant here not as a function of an individual software component, but as a system-level outcome of coordinated information acquisition, interpretation, and interaction with the human operator.

The role of architecture becomes more challenging when the knowledge used for this reasoning is itself dynamically constructed. Earlier knowledge-based systems relied predominantly on explicitly engineered and curated knowledge, such as rules, models, and expert knowledge prepared before deployment [5,17]. Runtime modelling subsequently introduced explicit representations that evolve together with the operating system [6]. Generative AI further changes this setting by combining runtime observations, retrieved documents, historical records, external information sources, and probabilistic model outputs during operation [3]. Consequently, operational knowledge may be heterogeneous, continuously evolving, and only partially trusted. Reasoning quality therefore depends not only on the reasoning mechanism itself but also on the completeness, consistency, provenance, temporal validity, and uncertainty of the evidence from which the operational interpretation is constructed [18, 19].

Trustworthy AI frameworks address related concerns through requirements and principles concerning transparency, robustness, human oversight, and gov-

ernance [20, 21]. These concerns are particularly important when the human operator retains authority over actions affecting physical systems. However, such principles do not by themselves determine which persistent architectural responsibilities are required to ensure that the operational knowledge presented to the operator has been adequately acquired, qualified, integrated, and justified. We use the term *trustworthy operational knowledge construction* to denote this continuous architectural process through which heterogeneous operational evidence is transformed into sufficiently justified, traceable, and contextually valid operational knowledge for human decision making.

This interpretation is consistent with the systems perspective that system-level qualities may emerge from coordinated interactions among multiple system elements rather than being attributable to a single component [14, 22]. Recent work similarly demonstrates that qualities such as ethics-aware autonomous behaviour may require explicit architectural support distributed across several architectural elements [23]. We therefore treat trustworthy operational knowledge construction as an architecturally significant, emergent concern when its realisation depends on coordinated architectural behaviour, influences system-level qualities and stakeholder concerns, and constrains subsequent architectural decisions [7,24–26]. Existing research provides the necessary concepts—situational awareness, runtime knowledge, trustworthy AI, emergent system qualities, and architectural responsibilities—but provides limited guidance for systematically connecting failures in operational knowledge construction to the architectural capabilities and responsibilities required to address them. The following section develops this connection through a failure-driven architectural reasoning process.

3 Failure-driven reasoning process (FDRP)

Failure-oriented engineering has traditionally focused on identifying hazards, unsafe interactions, and the mechanisms required to prevent unacceptable system behaviour [9]. Rather than analysing failures of software components, Leveson [9] argues that undesirable system behaviour frequently emerges from inadequate interactions between system elements and their operational environment. Following this systems-oriented perspective, this work shifts the focus towards failures occurring during the construction of operational knowledge and investigates how such failures can systematically inform architectural design. Failure-oriented engineering methods such as Systems-Theoretic Process Analysis (STPA) [9] and Failure Mode and Effect Analysis (FMEA) [27,28] have long been used to identify operational failures and derive mitigation strategies. The proposed failure-driven reasoning process (FDRP) is based on the assumption that operational decision-making in AI-enabled operational systems depends on the construction of an adequate operational situation. Following the established software architecture literature [7, 8, 29], we treat architecture as a structure of responsibilities, design decisions, and relationships that address stakeholder concerns and quality requirements. In this work, the primary stakeholder is the

human operator or field engineer who must make decisions under uncertainty.

The previous section argued that operational decision support in complex socio-technical systems differs fundamentally from traditional information processing. AI-enabled operational systems are no longer responsible merely for collecting, processing, and presenting operational data. Instead, they increasingly participate in constructing the operational knowledge upon which human situational awareness and decisions depend [16, 31]. This shift has important architectural consequences. While existing architectural approaches successfully organise software according to functional decomposition, communication mechanisms, deployment structures, or adaptation loops [7], they provide comparatively limited guidance for modelling how trustworthy operational knowledge should be systematically constructed from heterogeneous, uncertain, and continuously evolving information [32].

We therefore argue that the modelling abstraction itself should be reconsidered. Rather than organising architectures around implementation technologies or software components, architectures supporting operational decision-making should be organised around persistent architectural responsibilities. Here we define responsibility as a stable architectural obligation that continuously contributes to the construction of trustworthy operational knowledge independently of the underlying implementation technology. Consequently, responsibilities remain stable even when reasoning mechanisms, communication infrastructures, or software components evolve over time. This perspective is consistent with responsibility-driven design principles [26], where responsibilities are identified before implementation artefacts, and with software architecture principles that derive architectures from architecturally significant concerns rather than implementation structures. However, unlike traditional responsibility-driven design, which focuses on software objects and their interactions, the proposed approach elevates responsibilities to the architectural level. The objective is therefore not to decompose software into components, but to identify the persistent architectural capabilities that must exist to support trustworthy operational decision-making.

The theoretical basis for deriving the responsibilities combines three perspectives. First, situational awareness research explains that operators need to perceive relevant information, comprehend its meaning, and project possible future states before acting [10, 15]. Second, Models@Runtime research shows that software-intensive systems may maintain runtime representations of the managed system and its environment, enabling analysis and adaptation during execution [6]. Third, complex systems research emphasises that operational knowledge is often uncertain, dynamic, emergent system property and only partially observable [22, 33, 34]. Therefore, the system cannot simply present raw data to the operator; it must support the construction, validation, interpretation, and explanation of operational knowledge.

As already explained in the previous section, we interpret operational knowledge as an emergent system property rather than as a single software function. It emerges when several architectural responsibilities work together to support the operator's understanding of the current operational state. This interpreta-

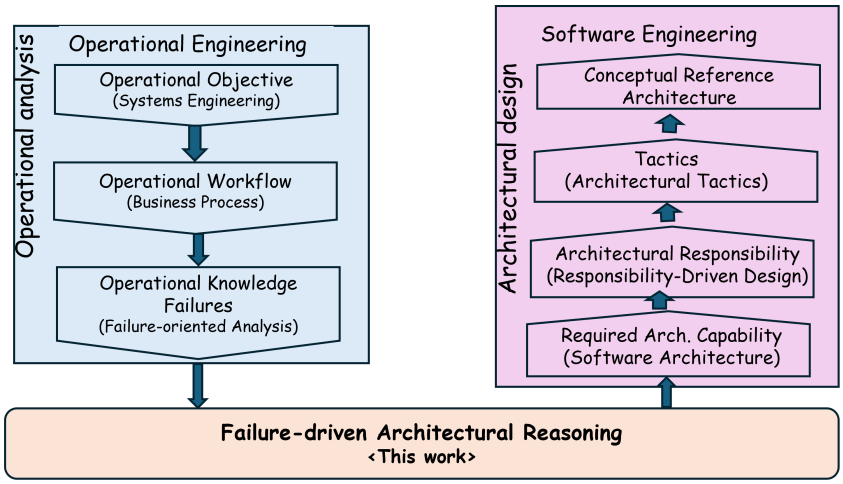

Figure 1: Conceptual framework of the proposed failure-driven architectural reasoning process. Existing engineering disciplines provide methods for analysing operational objectives, workflows, and operational knowledge failures (left), and methods for designing software architectures through capabilities, responsibilities, and architectural tactics (right). The proposed work contributes the reasoning process that bridges these two engineering domains by systematically transforming operational knowledge failures into architectural design decisions supporting trustworthy operational knowledge construction.

tion is also consistent with recent trustworthy AI and regulatory perspectives, which emphasize transparency, human oversight, traceability, and risk control in systems that support decisions in high-impact contexts [20, 21, 32].

Figure 1 summarises the conceptual FDRP proposed in this paper. Rather than representing the resulting software architecture, the figure illustrates the reasoning bridge that connects operational engineering with software architecture. The left-hand side represents concepts adopted from systems engineering, business process analysis, and failure-oriented engineering, while the right-hand side represents established software architecture concepts. The proposed contribution lies in the explicit reasoning process that systematically translates operational knowledge of failures into architectural capabilities, responsibilities, architectural tactics, and ultimately a conceptual reference architecture. The FDRP consists of six conceptual transitions connecting operational engineering with software architecture. Rather than prescribing a fixed architectural methodology, the process provides an explicit reasoning chain that systematically translates operational problems into architectural design decisions.

(1) Operational objective. Operational objectives define the desired operational outcome that software is expected to support and establish the context for subsequent workflow analysis.

(2) Operational workflow. Operational workflows describe the sequence of activities through which the operational objective is achieved and expose situations where operational knowledge may become unreliable.

(3) Failure-to-capability reasoning. Each operational knowledge failure implies one or more required architectural capabilities capable of preventing, detecting, or mitigating the identified deficiency. This transition constitutes the central contribution of the proposed FDRP because it establishes an explicit bridge between operational engineering and software architecture.

(4) Architectural capabilities are subsequently assigned to persistent architectural responsibilities. While a capability describes *what* the architecture must continuously provide, a responsibility identifies the architectural obligation responsible for ensuring that the capability remains available throughout system operation.

(5) Architectural responsibilities are implementation-independent and may therefore be realised through different architectural tactics depending on the target application domain, implementation technology, and architectural style.

(6) Finally, **architectural tactics** are organised into a conceptual reference architecture illustrating one possible architectural realisation of the derived responsibilities. The resulting architecture should therefore be interpreted as an outcome of the FDRP.

The proposed FDRP was not developed through an intuitive decomposition of software functionality. Instead, it was derived using a deductive architectural reasoning process that progressively constrains the architectural design space. The objective of this process is to ensure that every identified architectural responsibility can be explicitly justified through traceable architectural reasoning rather than subjective design decisions. The derivation begins by identifying the operational objective that the AI-assisted operational system is expected to support. Operational objectives describe the desired outcome of an operational activity independently of any implementation technology. Examples include reliable operational decision-making, safe execution of operational actions, or trustworthy interpretation of heterogeneous observations. Each operational objective exposes one or more architecturally significant concerns. For software architects, architecture principles and architectural concerns represent the quality properties that significantly influence architectural decisions [7, 29, 30]. In AI-enabled operational systems, these concerns typically include trustworthiness, contextual consistency, transparency, accountability, adaptability, and decision reliability [20, 21, 32]. Failure-oriented engineering approaches such as STPA argue that system-level failures should be analysed to identify the constraints required to prevent hazardous system behaviour [9]. Building upon this principle, we argue that failures related to operational knowledge construction can similarly reveal architectural capabilities that must be continuously provided by the software architecture in order to prevent unreliable operational reasoning. Therefore, in our approach for every identified concern, the derivation process analyses the operational conditions that may prevent the concern from being continuously satisfied, similarly to the FMEA derivation procedure [27,28].

These operational failure modes represent recurring situations in which the architecture may produce an incomplete, inconsistent, or misleading operational understanding. Typical examples include unreliable information sources, conflicting observations, missing contextual information, unsupported recommendations, or insufficient explanation for human operators.

Failure-oriented engineering methods identify what may prevent operational objectives from being achieved and derive the constraints required to mitigate undesirable system behaviour [9]. Systems engineering subsequently introduces capabilities as persistent system abilities required to achieve operational objectives under specified operational conditions [35, 36]. Finally, Responsibility-Driven Design interprets responsibilities as enduring obligations assigned to software elements to continuously realise such capabilities [26]. Building upon these complementary perspectives, the FDRP systematically transforms operational knowledge failures into architectural capabilities and subsequently into architectural responsibilities supporting trustworthy operational knowledge construction. The identification of failure modes naturally leads to the architectural capabilities required to prevent or mitigate these failures. Unlike software functions, these capabilities are persistent architectural properties that must continuously exist regardless of implementation technology. They describe what the architecture must always be capable of providing to maintain trustworthy operational knowledge construction. Finally, each required capability is realised through one or more architectural responsibilities. An architectural responsibility is therefore interpreted as a stable architectural obligation responsible for continuously providing a required operational capability. Responsibilities remain independent of implementation technologies and may subsequently be realised using different architectural styles, software components, services, autonomous agents, or artificial intelligence techniques.

An important characteristic of this derivation process is its traceability. Every architectural responsibility can be traced back through the capability it realises, the operational failure mode it mitigates, the architecturally significant concern it addresses, and ultimately the operational objective that motivated its existence. Conversely, every operational objective can be systematically decomposed into the architectural responsibilities required to support it. This bidirectional traceability provides a rationale for architectural decisions and reduces the subjectivity typically associated with conceptual architecture design.

The derivation process is inherently iterative. During architectural analysis, responsibilities are repeatedly refined, decomposed, merged, or separated as new evidence becomes available. Two candidate responsibilities are merged only if they address the same architecturally significant concern and mitigate the same class of operational failure modes. Conversely, a responsibility is decomposed whenever different concerns or failure mechanisms require independent architectural capabilities. The resulting responsibility model therefore represents the simplest set of responsibilities that collectively satisfies all identified architectural concerns while preserving complete traceability from operational objectives to architectural decisions.

This deductive reasoning process forms the methodological foundation of the

proposed responsibility-driven architectural modelling approach. The method first derives architecturally justified responsibilities and only subsequently organises them into a coherent reference architecture.

4 Illustrative Operational Scenario

This section introduces the operational scenario used to illustrate the architectural problem addressed by the proposed reasoning process. The objective is not to evaluate 5G deployment technologies themselves, but to expose an operational workflow in which AI-assisted software participates in constructing the operational knowledge upon which human decisions depend. The scenario therefore serves as a representative operational example through which failures in operational knowledge construction can be analysed before introducing the proposed FDRP.

4.1 The 5G-VINNI Deployment Case

The illustrative scenario is derived from the deployment experience reported by the 5G-VINNI (5G Verticals Innovation Infrastructure) project. 5G-VINNI was a large European initiative that established geographically distributed experimental 5G infrastructures for validating advanced network technologies and vertical applications under realistic operational conditions. The project documented practical engineering experience obtained during the project execution.

The deployment experience reported for the United Kingdom testbed was selected because it describes the complete operational workflow performed by field engineers during deployment and commissioning. The reported activities extend beyond the physical installation of radio equipment and include planning, coverage verification, interpretation of unexpected measurements, decision making, and operational adaptation [37].

Although originating from telecommunications, the analysed workflow represents a broader class of AI-assisted operational processes encountered in critical infrastructure domains. Similar operational reasoning activities appear whenever human operators continuously integrate heterogeneous observations, engineering knowledge, runtime information, and AI-generated recommendations before deciding whether physical interventions are required.

4.2 Operational Workflow

Based on the deployment experience reported by the 5G-VINNI project and subsequently reviewed against industrial deployment practice at Elektronika Nova Ltd., Pula, Croatia, the analysed workflow consists of six recurring operational activities, listed in the Table 1:

1. Site survey and deployment planning;
2. Installation and system configuration;

Table 1: Failure-oriented analysis of operational knowledge construction in the illustrative 5G-VINNI deployment workflow

Workflow Activity	**Required Operational Knowledge**	**Operational Knowledge Failure**	**Possible Cause**	**Operational Consequence**
Site survey and planning	Deployment constraints, environmental context, expected coverage	Incomplete operational context	Missing environmental information, outdated planning assumptions	Incorrect deployment assumptions
Installation and configuration	Current deployment state and configuration	Configuration mismatch	Provisioning errors, incomplete topology discovery	Incorrect diagnosis of deployment problems
Coverage verification	Correct interpretation of field measurements	Incorrect interpretation of observations	Reflection, shadowing, incomplete context	Wrong corrective action
Operational interpretation	Integrated operational understanding	Unsupported operational explanation	Single hypothesis, insufficient evidence	Incorrect operational recommendation
Decision making	Justified operational recommendation	Recommendation not sufficiently justified	Missing evidence, low confidence, conflicting observations	Incorrect field intervention
Operational adaptation	Updated operational knowledge	Operational learning not preserved	Missing feedback integration	Repeated operational errors

3. Coverage verification;
4. Interpretation of operational observations;
5. Corrective decision making;
6. Operational adaptation.

The workflow illustrates the continuous construction and refinement of operational knowledge throughout deployment. Engineers repeatedly acquire new evidence, revise deployment assumptions, interpret heterogeneous observations, and update their operational understanding before deployment decisions are accepted.

4.3 Operational Knowledge Failure Analysis

Following a failure-oriented engineering perspective, the analysed workflow was examined to identify situations in which incomplete, inconsistent, outdated, or insufficiently justified operational knowledge may lead to incorrect operational decisions. Unlike traditional failure analysis, the objective here is not to analyse failures of physical or software components, but failures emerging during operational knowledge construction.

Table 1 summarises the resulting analysis. Each workflow activity is interpreted as a knowledge-producing function whose purpose is to progressively construct the operational understanding required for deployment decisions. The table identifies the operational knowledge required at each activity together with the corresponding knowledge failure, its possible causes, and its operational consequences.

An important observation emerging from the analysis is that these failures do not primarily originate from software malfunction. Instead, they arise when operational knowledge becomes incomplete, inconsistent, fragmented, outdated, or insufficiently justified. Consequently, incorrect operational decisions may occur even when software components operate correctly. The analysed failures should therefore be interpreted as operational knowledge failures rather than software failures.

4.4 Architectural Implications

The identified operational knowledge failures reveal a common architectural consequence. Every failure implies that the software architecture must continuously provide one or more capabilities capable of preventing, detecting, or mitigating the identified deficiency before operational recommendations are presented to human operators. Note that mapping between operational knowledge failures and architectural responsibilities is not necessarily one-to-one. A single knowledge failure may require several complementary architectural capabilities and, consequently, multiple architectural responsibilities addressing different aspects of the same failure.

Table 2 summarises the traceability between the identified knowledge failures and the architectural capabilities required to address them. The table does not yet prescribe software components or implementation technologies. Instead, it establishes an explicit connection between operational failures and the persistent architectural capabilities required to support trustworthy operational knowledge construction.

Existing software architecture, failure-oriented engineering, human factors, and trustworthy AI research each contribute concepts required to understand different parts of this reasoning process. Nevertheless, these approaches provide only limited guidance for systematically transforming operational knowledge failures into architectural capabilities and subsequently into architectural responsibilities. This observation motivates the need for the FDRP introduced in this paper.

Table 2: Traceability from operational knowledge failures to architectural responsibilities

Operational Knowledge Failure	**Required Architectural Capability**	**Derived Architectural Responsibility**
Incomplete operational context	Acquire and synchronise relevant operational evidence; assess its provenance, freshness, and reliability	Context Acquisition; Trust Assessment
Configuration mismatch	Maintain an explicit and current representation of the actual runtime configuration and operational state	Runtime Operational Knowledge Management
Incorrect interpretation of observations	Detect inconsistencies among heterogeneous observations and integrate available evidence into a coherent operational representation	Context Validation; Context Integration
Unsupported operational explanation	Generate and evaluate alternative explanations using available operational evidence	Operational Reasoning
Recommendation not sufficiently justified	Assess evidence sufficiency, uncertainty, constraints, and expected consequences; expose the recommendation rationale and support human assessment	Decision Assurance; Intent and Explanation Generation; Human Deliberation Support
Operational learning not preserved	Capture outcomes of operational actions and incorporate them into evolving operational knowledge	Knowledge Evolution

5 Discussion

The objective of this study is to define the trustworthy operational knowledge construction problem, propose a failure-driven architectural derivation method, and illustrate and investigate its applicability in a realistic operational scenario. The proposed method differs from most existing software architectures by shifting the primary modelling abstraction from software components to architectural responsibilities. Existing reference architectures generally decompose systems according to functional modules, deployment layers, services, or communication mechanisms. In contrast, the proposed approach decomposes the

architecture according to persistent responsibilities that are required to support trustworthy operational knowledge construction. Existing approaches primarily address complementary concerns and therefore leave important architectural questions unanswered when software actively participates in constructing operational knowledge. This distinction is particularly important in operational environments where multiple technologies may evolve independently. For example, adaptive reasoning may be implemented using rule engines, knowledge graphs, Bayesian models, or Large Language Models without affecting the remaining architectural structure. Consequently, the proposed model separates architectural concerns from implementation technologies, allowing the architecture to remain stable despite rapid technological evolution.

Rather than treating situational awareness as an isolated software function, the proposed architecture interprets it as an emergent system property produced through the coordinated execution of multiple architectural responsibilities. Under this interpretation, situational awareness cannot be assigned to a single architectural layer but emerges from the interaction between context acquisition, trust assessment, runtime operational knowledge management, context validation, context integration, operational reasoning, decision assurance, intent and explanation generation, human deliberation support, and knowledge evolution. The architectural failure analysis further illustrates that incorrect operational decisions are not necessarily caused by failures of software components or AI models themselves. Instead, they frequently originate from failures in the construction of operational knowledge, including incomplete contextual information, inconsistent observations, outdated engineering knowledge, unsupported interpretations, or insufficiently justified recommendations. Such failures may ultimately lead to repeated field interventions, increased operational costs, unnecessary exposure of engineers to hazardous working conditions, or reduced quality of the deployed infrastructure.

The study is intentionally positioned as preliminary research. Rather than proposing a complete requirements engineering methodology, it investigates whether the proposed derivation process produces architectural abstractions that are meaningful to domain experts and sufficiently expressive to guide subsequent architectural design. The deployment and commissioning of 5G radio infrastructure was selected as the validation scenario because it exhibits several characteristics representative of contemporary AI-assisted operational systems. The workflow requires continuous integration of heterogeneous information sources, including deployment plans, environmental observations, radio signal measurements, equipment configuration, engineering documentation, and technician expertise. Operational decisions are made under uncertainty, frequently require iterative refinement, and directly affect both deployment quality and operator safety. Furthermore, the selected scenario represents a realistic industrial workflow. The operational activities were reconstructed from the deployment experience reported in the 5G-VINNI project and subsequently reviewed against practical experience obtained during commercial 5G antenna installation and commissioning activities performed at Elektronika Nova Ltd. This combination of published operational evidence and industrial practice pro-

vides sufficient confidence that the analysed workflow represents typical field deployment activities rather than project-specific procedures.

The 5G-VINNI workflow analysis identified six principal operational activities: site survey and planning, installation and configuration, coverage verification, operational interpretation, decision making and operational adaptation. Here, we are not considering these activities as software functions, but they were interpreted as operational tasks performed by field engineers. Each activity was analysed to identify the operational concern that limits trustworthy operational knowledge construction. The operational workflow analysed demonstrates that AI-assisted operational decision-making is fundamentally a knowledge construction process. Before an operational recommendation can be accepted, heterogeneous observations originating from measurements, deployment plans, engineering documentation, environmental conditions, historical experience, and runtime system state must be continuously transformed into a coherent and trustworthy operational understanding.

Although demonstrated through a 5G installation scenario, the proposed responsibilities are intentionally domain-independent. The individual responsibilities describe architectural capabilities that appear whenever human operators must make decisions using heterogeneous and uncertain information. For example, in manufacturing environments, the same responsibilities may integrate production telemetry, machine diagnostics, maintenance records, and operator observations before generating production recommendations. In water distribution systems, sensor measurements, hydraulic simulations, weather forecasts, and maintenance information must similarly be integrated before supporting operational decisions regarding leakage detection, valve control, or emergency response. Although the operational information differs across domains, the architectural responsibilities remain largely unchanged. This observation suggests that the proposed model represents a reusable architectural concept rather than a domain-specific solution. Future work should therefore investigate its applicability across multiple operational domains to evaluate its generalisability.

This work represents an initial conceptual contribution and therefore has several limitations. First, the proposed responsibilities have been derived from architectural analysis and demonstrated through a single operational case study. Although the identified responsibilities are grounded in existing software architecture, human factors, and self-adaptive systems literature, additional empirical validation across multiple domains is required. Second, the proposed architecture intentionally remains technology-independent. Consequently, the paper does not evaluate alternative implementations of individual responsibilities, nor does it compare the performance of different reasoning mechanisms such as rule-based systems, knowledge graphs, or Large Language Models.

Finally, this work focuses on architectural modelling rather than quantitative evaluation. Future research should therefore investigate measurable architectural quality attributes associated with operational situation construction, including trustworthiness, explainability, contextual consistency, and decision reliability.

6 Conclusion

This paper formulated trustworthy operational knowledge construction as an emerging software architecture concern introduced by AI-assisted operational decision making. By connecting operational knowledge failures with architectural capabilities and responsibilities through a failure-driven reasoning process, the paper establishes a conceptual foundation for future research on software architectures supporting AI-enabled critical infrastructures. Future work should investigate implementation-specific reference architectures and empirical evaluation methods capable of assessing the effectiveness of the proposed FDRP across different operational domains.

References

[1] R. Rajkumar, I. Lee, L. Sha and J. Stankovic, "Cyber-physical systems: The next computing revolution," Design Automation Conference, Anaheim, CA, USA, 2010, 731-736.

[2] X. Hou et al., Large Language Models for Software Engineering: A Systematic Literature Review. ACM Trans. Softw. Eng. Methodol. 33, 8, Article 220, 79 pages, 2024.

[3] P Lewis et al., "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks" NIPS'20: Proceedings of the 34th International Conference on Neural Information Processing Systems, Article No.: 793, Pages 9459 - 9474.

[4] J. Li et al., Generative AI for Self-Adaptive Systems: State of the Art and Research Roadmap. ACM Trans. Auton. Adapt. Syst. 19, 3, Article 13, 60 pages, 2024.

[5] R. Studer, V.R. Benjamins, D. Fensel, Knowledge engineering: Principles and methods, Data and Knowledge Engineering, Vol. 25, Issues 1–2, 1998, 161-197.

[6] N. Bencomo, S. Götz, H. Song, "Models@run.time: A Guided Tour of the State of the Art and Research Challenges," *Software and Systems Modeling*, vol. 18, no. 5, pp. 3049–3082, 2019.

[7] H. van Vliet, *Software Engineering: Principles and Practice*, 3rd ed., John Wiley & Sons, 2008.

[8] S. Angelov, P. Grefen, D. Greefhorst, "A Framework for Analysis and Design of Software Reference Architectures," *Information and Software Technology*, vol. 54, no. 4, 417–431, 2012.

[9] N. G. Leveson, *Engineering a Safer World: Systems Thinking Applied to Safety*, MIT Press, 2011.

[10] M. R. Endsley, Toward a Theory of Situation Awareness in Dynamic Systems, *Human Factors*, vol. 37, no. 1, pp. 32–64, 1995.

[11] J. D. Lee and K. A. See, "Trust in Automation: Designing for Appropriate Reliance," *Human Factors: The Journal of the Human Factors and Ergonomics Society*, vol. 46, no. 1, pp. 50–80, 2004.

[12] K. A. Hoff and M. Bashir, "Trust in Automation: Integrating Empirical Evidence on Factors That Influence Trust," *Human Factors: The Journal of the Human Factors and Ergonomics Society*, vol. 57, no. 3, pp. 407–434, 2015.

[13] E. Glikson and A. W. Woolley, "Human Trust in Artificial Intelligence: Review of Empirical Research," *Academy of Management Annals*, vol. 14, no. 2, pp. 627–660, 2020.

[14] Hollnagel, E., Woods, D. D., Leveson, N. (Eds.). (2006). Resilience engineering: Concepts and precepts. Ashgate Publishing.

[15] M. Endsley and D. Jones, Designing for Situation Awareness (Second ed.). CRC Press. p. 13. 2016.

[16] R Fang et al., "A Systematic Review of Empirical Studies on Situation Awareness: Perspectives From the Interaction Among Humans, Machines, and the Task Environment." Psych J. 2025, 14(5):718-733.

[17] Edward A. Feigenbaum. "The Art of Artificial Intelligence: Themes and Case Studies of Knowledge Engineering." Proceedings of IJCAI, 1977.

[18] E. Miedema, S. Waschull, and C. Emmanouilidis, "Towards trustworthy artificial intelligence for decision-making: A lifecycle perspective on knowledge- and data-driven artificial intelligence systems," *Computers in Industry*, vol. 174, p. 104409, 2026.

[19] B. Li, P. Qi, B. Zhou, et al., "Trustworthy AI: From Principles to Practices," *ACM Computing Surveys*, vol. 55, no. 9, pp. 1–46, Jan. 2023.

[20] National Institute of Standards and Technology, *Artificial Intelligence Risk Management Framework (AI RMF 1.0)*, NIST AI 100-1, Gaithersburg, MD, 2023.

[21] European Union, "Regulation (EU) 2024/1689 of the European Parliament and of the Council laying down harmonised rules on Artificial Intelligence (Artificial Intelligence Act),"Official Journal of the European Union, 2024.

[22] Y. Bar-Yam, *Dynamics of Complex Systems*, Addison-Wesley, 1997.

[23] M. Autili at all., "A reference architecture for ethical-aware autonomous systems." J. Syst. Softw. Vol. 235, 2026, 112749.

[24] P. Kruchten, R. Capilla, J. C. Dueñas, "The Decision View's Role in Software Architecture Practice," *IEEE Software*, vol. 26, no. 2, pp. 36–42, 2009.

[25] Jeff Tyree and Art Akerman. 2005. Architecture Decisions: Demystifying Architecture. IEEE Softw. 22, 2 (March 2005), 19–27.

[26] R. Wirfs-Brock, B. Wilkerson, L. Wiener, *Designing Object-Oriented Software*, Prentice Hall, 1990.

[27] MIL-STD-1629A, *Procedures for Performing a Failure Mode, Effects and Criticality Analysis*, U.S. Department of Defense, 1980 (reprinted 1993).

[28] D. H. Stamatis, *Failure Mode and Effect Analysis: FMEA from Theory to Execution*, 2nd ed., ASQ Quality Press, 2003.

[29] ISO/IEC/IEEE, *ISO/IEC/IEEE 42010:2022 – Systems and Software Engineering – Architecture Description*, International Organization for Standardization (ISO), International Electrotechnical Commission (IEC), Institute of Electrical and Electronics Engineers (IEEE), Geneva, Switzerland, 2022.

[30] Bass, L., Clements, P., Kazman, R. (2021). Software architecture in practice (4th ed.). Addison-Wesley Professional.

[31] Li J, Zhou ZC, Wang ZC, Lv H. Prioritizing human-AI collaboration in healthcare: the TRIAD framework for trustworthy governance, real-world, and integrated adaptive deployment. Mil Med Res. 2026, 12:97.

[32] N. Díaz-Rodríguez at all., "Connecting the Dots in Trustworthy Artificial Intelligence: From AI Principles, Ethics, and Key Requirements to Responsible AI Systems and Regulation," *Information Fusion*, vol. 99, Art. 101896, 2023.

[33] P. Cilliers, *Complexity and Postmodernism: Understanding Complex Systems*, Routledge, London, 1998.

[34] J. H. Holland, *Complexity: A Very Short Introduction*, Oxford University Press, Oxford, 2014.

[35] ISO/IEC/IEEE, *ISO/IEC/IEEE 15288:2023 – Systems and Software Engineering – System Life Cycle Processes*, International Organization for Standardization (ISO), International Electrotechnical Commission (IEC), Institute of Electrical and Electronics Engineers (IEEE), Geneva, Switzerland, 2023.

[36] INCOSE, *Systems Engineering Handbook: A Guide for System Life Cycle Processes and Activities*, 4th ed., John Wiley & Sons, 2015.

[37] M. Ghassemian, P. Muschamp and D. Warren, "Experience Building a 5G Testbed Platform," 2020 IEEE 3rd 5G World Forum (5GWF), Bangalore, India, 2020, pp. 473-478.