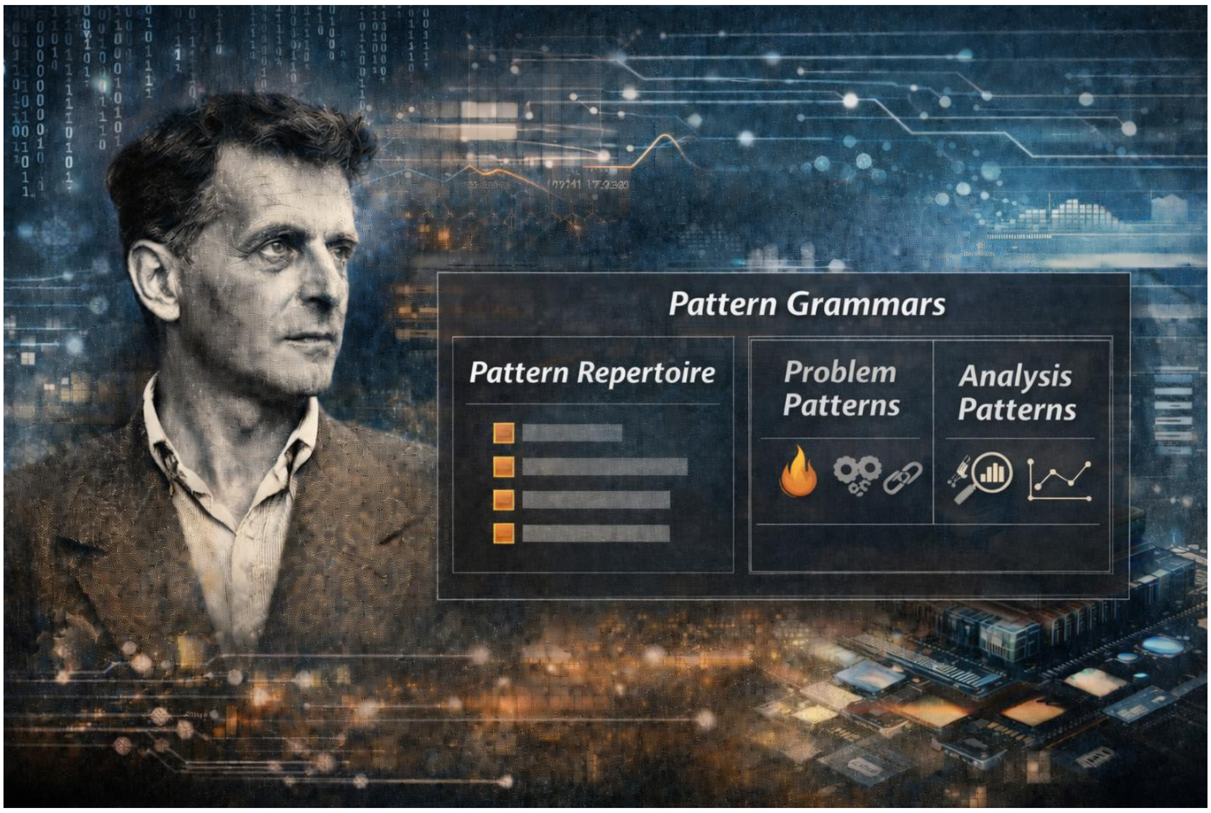

Pattern-Oriented Diagnostics and Observability as a Philosophy of Engineering: From Correspondence to Grammar, from Problem Patterns to Analysis Patterns

Introduction

Every mature engineering discipline rests on philosophical assumptions, whether or not they are made explicit. In software engineering, diagnostics and observability have traditionally assumed that artefacts produced by a running system directly represent what the system is doing. Correct diagnosis, on this view, consists in reading those artefacts accurately, while observability is understood as increasing the quantity and fidelity of such representations. Together, diagnostics and observability are treated as epistemic extensions of execution itself.

As systems evolved toward massive concurrency, distribution, adaptive behaviour, and ubiquitous instrumentation, diagnostic and observability artefacts ceased to form a single coherent picture. Engineers increasingly encountered situations in which logs contradicted traces, metrics obscured rather than clarified behaviour, and different observability pipelines suggested incompatible explanations. In practice, engineers responded by developing extensive diagnostic pattern repertoires[1] that encode accumulated interpretive knowledge. Philosophically, however, these developments expose the breakdown of an implicit correspondence theory shared by both diagnostics and observability.

[1] Pattern Repertoire, Theoretical Software Diagnostics, Fourth Edition, page 229

Foundations

Pattern-Oriented Diagnostics and Observability rests on a small set of foundational commitments that shape how execution, artefacts, and interpretation are understood. These commitments are not methodological preferences, but conceptual necessities imposed by the nature of modern software systems.

First, execution and observation are fundamentally distinct. Execution unfolds as a concrete, time-bound process governed by scheduling, resource contention, concurrency, and environmental interaction. Observability artefacts, such as logs, traces, memory dumps, metrics, and derived signals, are not neutral windows onto this process. They are products of instrumentation, sampling, aggregation, and representation choices. As such, they are already interpretations before any human analysis begins.

Second, diagnostic meaning is not intrinsic to artefacts. Artefacts do not carry determinate meanings independent of their use. Their significance arises only within diagnostic practice, shaped by questions being asked, transformations applied, and comparisons performed. This rejects the assumption that artefacts "speak for themselves" and replaces it with the view that meaning is established through disciplined interpretive activity[2].

Third, patterns are primary units of diagnostic reasoning. Rather than reasoning directly from artefacts to causes, diagnosticians reason through patterns that stabilise interpretation. Problem patterns capture recurring structures in execution failure, while analysis patterns define how artefacts are rendered intelligible in the first place[3]. This distinction is foundational: without analysis patterns, problem patterns cannot be reliably recognised.

Fourth, diagnostic reasoning is inherently sequential and contextual. Patterns often arise in succession[4], where earlier patterns enable or block later ones. Understanding failures, therefore, requires reconstructing ordered diagnostic narratives rather than identifying isolated symptoms. Causality in diagnostics is temporal and grammatical, not merely local.

Finally, judgment is irreducible. No combination of artefacts, patterns, notations, or tools can eliminate the need for expert judgment. Tooling can support rule-following, but it cannot replace it. Diagnostics and observability are thus practices grounded in shared grammar, training, and experience rather than in purely mechanical inference.

[2] Philosophy of Software Diagnostics: An Introduction, Part 1

[3] Pattern! What Pattern? Theoretical Software Diagnostics, Fourth Edition, 216

[4] Pattern Succession, Ibid., page 81

These foundations establish diagnostics and observability as interpretive engineering disciplines. They motivate the shift from correspondence to grammar and frame Pattern-Oriented Diagnostics and Observability as a coherent philosophy of engineering rather than a collection of techniques.

The Tractarian Phase of Diagnostics and Observability

In the *Tractatus Logico-Philosophicus*[5], the philosopher Ludwig Wittgenstein[6] proposes that propositions are pictures of facts that share their logical form. Classical diagnostics and early observability implicitly adopt this picture theory. Execution is treated as a set of objective facts, and artefacts are taken to be propositional representations of those facts. A crash dump is read as stating the cause of failure, a trace as revealing causal order, and a metric as measuring a real quantity in the system.

Within this regime, patterns are understood primarily as descriptions of recurring execution phenomena. Memory leaks, deadlocks, race conditions, resource exhaustion, and similar failures are catalogued as problem patterns: recognisable forms of breakdown in execution behaviour. Observability is assumed to make these patterns visible by providing increasingly detailed pictures of execution. This view corresponds to the Tractarian phase of engineering diagnostics and observability, in which meaning is understood to arise from structural correspondence between the artefact and its execution.

The Breakdown of Correspondence under Observability

Modern systems systematically violate the assumptions that underpin correspondence. Instrumentation perturbs the behaviour it observes, introducing bias and distortion. Sampling, aggregation, and retention policies reshape artefacts before they are ever inspected. Artefacts produced by different observability pipelines frequently disagree, with no principled way to determine which representation should be privileged. In distributed systems, causality is fragmented across time, space, and administrative boundaries. In systems that incorporate machine learning, internal states such as embeddings and activations are not propositional and cannot meaningfully be said to represent execution facts. In other words, ML internal states do not function like statements about the system or the world in the way logs, traces, or variable values traditionally do.

In these conditions, recognising a familiar problem pattern is no longer sufficient. The same apparent symptom may arise from execution behaviour, from measurement artefacts, or from the grammar imposed by observability itself. Artefacts no longer determine their own interpretation. This situation mirrors the philosophical crisis that

[5] https://en.wikipedia.org/wiki/Tractatus_Logico-Philosophicus
[6] https://en.wikipedia.org/wiki/Ludwig_Wittgenstein

led Wittgenstein to abandon picture theory: propositions, like observability artefacts, cannot determine their own use.

The Turn to Practice in Diagnostics and Observability

In *Philosophical Investigations*[7], Wittgenstein replaces correspondence with a theory of meaning grounded in use. Understanding becomes participation in language games, rule-following becomes a practice rather than an algorithm, and categories are constituted by family resemblance rather than strict definition. Meaning is no longer located in a relation between sign and fact, but in the way signs are employed within shared forms of life.

Applied to engineering, this implies that diagnostics and observability are not passive epistemic activities but active interpretive practices. Artefacts acquire meaning only through their use in diagnostic contexts. The same trace, metric, or memory snapshot may support different explanations depending on the investigative question, the analysis performed, and the diagnostic language game being played. In practice, this use-based understanding is embodied in diagnostic pattern repertoires.

Problem Patterns and Analysis Patterns

Pattern-Oriented Diagnostics and Observability introduces a crucial distinction between two classes of patterns that are often conflated. Problem patterns characterise recurring forms of failure or anomalous behaviour in execution. They describe what kinds of things tend to go wrong in systems and provide a vocabulary for naming such situations. These patterns are extensively documented in diagnostic pattern repertoires developed through decades of practice, spanning crash, memory, trace, log, and metric domains. For example, detailed catalogues of concrete memory and trace patterns exist that illustrate hundreds of identifiable execution-level conditions[8].

Analysis patterns, by contrast, do not describe failures. They govern how artefacts produced by diagnostics and observability are interpreted. They specify how memory dumps are abstracted into structural regions, how traces are segmented into activity sequences, how logs are correlated or anchored, and how metrics are normalised or compared. Recognising a problem pattern presupposes that artefacts have already been rendered intelligible through appropriate analysis patterns.

Crucially, problem patterns rarely occur in isolation. In practice, they often appear in succession, where one pattern creates the structural or temporal conditions for another to emerge. Heap corruption may trigger a hard error, blocking execution and producing thread wait chains that may, in turn, propagate across process boundaries. Diagnosing

[7] https://en.wikipedia.org/wiki/Philosophical_Investigations
[8] DumpAnalysis.org + TraceAnalysis.org

such failures, therefore, requires tracing not just individual patterns, but their ordered succession across execution time.

Analysis Patterns as Diagnostic and Observability Grammar

Analysis patterns function as grammar in the Wittgensteinian sense. They do not assert facts about execution. Instead, they constrain what counts as a meaningful diagnostic or observability statement. An analysis pattern defines which artefacts are relevant, which transformations are legitimate, which comparisons are admissible, and which forms of counter-evidence are decisive.

Because analysis patterns operate grammatically, they admit symbolic and graphical representations. Symbolic notation[9] facilitates compact linguistic expressions of pattern combinations, using capitalised letters for major categories and subscripts for subcategories. This notation forms an explicit grammar of pattern composition.

Complementing symbolic forms, graphical notation has been developed to convey pattern structure and interpretation visually. The *Dia|gram graphical diagnostic analysis language*[10] provides a diagrammatic syntax that represents execution state, artefacts, and pattern relationships in visual form. It illustrates memory, trace, and log analysis patterns using a consistent visual grammar that supports comparison, sequencing, and layered interpretation. Graphical patterns facilitate communication of how patterns are distributed, related, and composed across artefacts, and serve as a bridge between conceptual grammar and practical analysis.

Within this combined grammatical framework, the succession of patterns becomes intelligible both textually and visually. Analysis patterns enable recognition not only of multiple problem patterns but also of how they relate in time and space, how observability artefacts reflect transformations in execution structure, and how diagnostic narratives emerge from artefact interactions.

Rule-Following, Judgment, and Tooling

Wittgenstein's rule-following considerations show that no rule contains the criteria for its own correct application. Following a rule is not a mechanical act but a practice sustained by shared standards, training, and judgment. This insight maps directly onto diagnostics and observability. Neither analysis patterns nor their symbolic or graphical notations determine their own use. They guide interpretation, but they do not replace it.

Tools can implement transformations, generate visualisations, or suggest candidate patterns. Still, they cannot determine when an analysis pattern is appropriate, when it should be abandoned, or when competing interpretations must be weighed. This limitation is particularly evident in complex incidents in which multiple pattern

[9] Notation for Memory and Trace Analysis, Theoretical Software Diagnostics, Fourth Edition, page 107
[10] Dia|gram Graphical Diagnostic Analysis Language, Ibid., page 251

successions are possible. Determining which pattern initiated a cascade, which patterns are consequences rather than causes, and which are artefacts of observation requires judgment that exceeds syntactic manipulation.

Pattern-Oriented Diagnostics and Observability does not treat this reliance on judgment as a weakness to be eliminated through automation. Instead, it recognises judgment as a structural feature of diagnostic practice. Tooling is therefore best understood as supporting rule-following practice rather than replacing it. Making analysis patterns explicit clarifies where judgment is exercised and why it cannot be fully mechanised.

Family Resemblance across Patterns and Observability Regimes

Problem patterns and analysis patterns do not form strict taxonomies with necessary and sufficient conditions. Instead, they exhibit family resemblance. Memory leaks across different runtimes share commonalities but do not have identical structures. Deadlocks, livelocks, and wait chains form clusters of related phenomena rather than sharply bounded categories. Analysis patterns likewise recur across domains with variation rather than uniformity[11].

This family-resemblance structure extends to pattern succession. Certain sequences of patterns recur frequently and become recognisable diagnostic trajectories, while others appear only in specific environments or under particular workloads. These successions cannot be reduced to universal causal laws, yet they are stable enough to guide expert reasoning. Their stability lies in shared diagnostic roles rather than in invariant mechanisms.

Understanding patterns through family resemblance explains how diagnostic knowledge transfers across platforms, languages, and observability stacks. What transfers is not a fixed mapping from artefact to cause, but a repertoire of analysis patterns and pattern successions that can be re-applied and adapted. Pattern-Oriented Diagnostics and Observability, therefore, treats the pattern repertoire as a living grammar rather than a closed classification system.

Second-Order Diagnostics and Observability

Because observability systems actively shape artefacts, they also shape interpretation. Sampling rates, aggregation windows, retention policies, and visualisation choices impose grammatical constraints on what can be seen and said about execution. As a result, observability itself becomes a legitimate object of diagnosis.

The second-order diagnostics addresses this reflexive layer. It asks not only what patterns appear in execution, but why certain patterns are visible while others are obscured. Pattern succession is particularly vulnerable to distortion at this level. Later

[11] Existential Prognostics: Periodic Table of Diagnostic Patterns, Ibid., page 273

patterns may be amplified by observability tooling, while earlier enabling patterns are suppressed or erased, leading diagnosticians to mistake consequences for causes.

Graphical and symbolic notation play a crucial role here by allowing comparison of analyses rather than artefacts alone. Differences in notational structure or diagrammatic composition can reveal differences in interpretive framing rather than differences in execution. Second-order analysis patterns thus make explicit a dimension of diagnostic practice that is otherwise tacit and error-prone.

AI as the Forcing Case

Artificial intelligence systems make explicit what was already implicit in complex software systems: the collapse of correspondence as a foundation for diagnostics and observability. In classical systems, artefacts could plausibly be treated as propositional representations of execution state. In AI systems, this assumption fails outright. Internal states such as embeddings, attention distributions, activation tensors, and learned weights do not stand in a picture relation to execution facts in any meaningful sense. They are not propositions, nor are they interpretable as such.

Observability in AI systems is therefore irreducibly indirect. Metrics, traces, and internal signals acquire meaning only through comparative, statistical, and behavioural interpretation. Explanations are rarely causal in a local sense; instead, they emerge from patterns observed across runs, datasets, training phases, or model variants. As a result, analysis patterns precede problem patterns even more strongly than in traditional systems. Without explicit analysis patterns, such as baselining, behavioural comparison, drift detection, or representational clustering, AI artefacts remain uninterpretable noise.

Pattern succession also becomes more prominent in AI diagnostics. In systems driven by learned representations, where *world models* and internal mappings replace simple symbolic states, behavioural anomalies often trigger secondary effects such as feedback loops, saturation, or representational collapse, producing sequences of diagnostic patterns rather than isolated failures. Comparing models by structure and stability rather than by performance alone is precisely what makes succession visible: a pattern that initially perturbs an internal representation may restructure latent spaces, lead to further cascading behaviours, or reshape the topology of model responses until the system reaches a stable, but pathological state where no further diagnostically meaningful variation is generated. This structural and topological view of internal model behaviour, and the need to treat sequences of patterns as diagnostic objects, is central to pattern-oriented observability in world models[12].

[12] Pattern-Oriented Diagnostics and Observability of World Models: A Topological Perspective (https://www.dumpanalysis.org/pattern-oriented-diagnostics-world-models-topological-perspective)

AI systems thus act as a forcing case for Pattern-Oriented Diagnostics and Observability. They demonstrate that diagnostics cannot be grounded in correspondence semantics; rather, they must be understood as a grammar-governed interpretive practice in which meaning arises from structured analysis, comparison, and rule-following under extreme opacity.

Pattern-Oriented Diagnostics and Observability as a Philosophy of Engineering

The philosophical position advanced here can now be stated with precision. Execution exhibits recurring structures that give rise to problem patterns, often unfolding in structured succession. Diagnostics and observability, however, are not governed by execution structure alone. They are governed by analysis patterns that function as grammar, constraining how artefacts may be produced, transformed, interpreted, ordered, and communicated.

This grammar is expressed through multiple representational forms: narrative explanation, symbolic notation, and graphical diagrams. None of these representations presents the facts of execution directly. Instead, they stabilise meaning within diagnostic practice. Execution obeys structure; diagnostics and observability obey practice.

By integrating problem patterns, analysis patterns, pattern succession, and explicit grammatical representations, Pattern-Oriented Diagnostics and Observability completes the philosophical arc from early to late Wittgenstein within engineering. It preserves correspondence where it works, but situates it within a broader framework in which meaning arises from rule-following, use, and disciplined interpretation under partial observability.

Related Work

This work intersects with several bodies of research across philosophy, software engineering, and systems diagnostics while remaining distinct in scope and emphasis.

In philosophy, it builds explicitly on the transition from early to late Wittgenstein, particularly the shift from picture theory to use-based meaning. While Wittgenstein's later philosophy has been applied extensively to language, mathematics, and social practices, its systematic application to engineering diagnostics and observability has received little attention.

In software engineering, debugging and observability literature has traditionally focused on tools, techniques, and heuristics. Patterns are typically treated as empirical best practices rather than as grammatical structures that govern interpretation. This paper departs from that tradition by reclassifying analysis patterns and their notation as grammar rather than heuristics.

Observability research often emphasises signal correlation and visualisation while underexamining how tooling shapes interpretation. Recent discussions of observability bias address related issues but stop short of a philosophical account of observability as grammar-producing infrastructure.

In AI diagnostics, interpretability and explainability research introduces new metrics and visualisations but struggles with the absence of correspondence semantics. The grammar-based approach proposed here offers a complementary perspective grounded in analysis patterns, their notation, and rule-following practice.

Conclusion

As software systems outgrew the assumptions of classical debugging and naive observability, the philosophical foundations of engineering diagnostics became visible. Pattern-Oriented Diagnostics and Observability provides a coherent response by distinguishing between patterns of failure and patterns of analysis, and by grounding both in a Wittgensteinian account of grammar, rule-following, and use. The succession of patterns observed in real systems is thereby understood not as a collection of isolated symptoms, but as a structured diagnostic narrative governed by explicit grammatical, notational, and diagrammatic constraints. Diagnostics and observability emerge not as mere data collection or decoding, but as disciplined interpretive practices under conditions of partial observability.