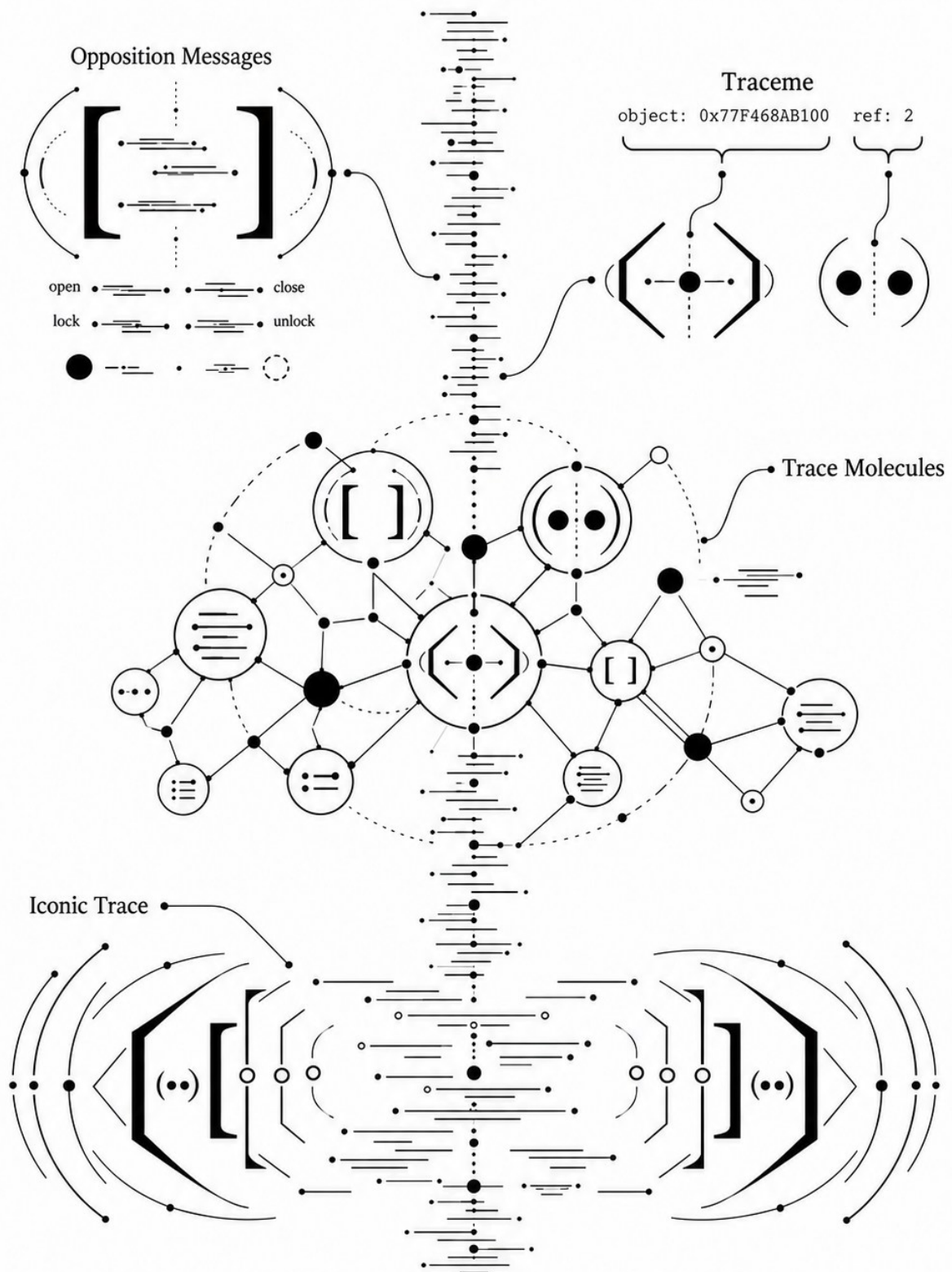


Semiotics in My Work on Software Diagnostics



Dmitry Vostokov, Software Diagnostics Institute

Semiotics in My Work on Software Diagnostics

From memiotics to observability and Codesign

Dmitry Vostokov

Developed with AI assistance (GPT-6 Astra and Claude Fable 5.1)

Version 12 | 17 October 2026

Abstract

I have used semiotics in my work on software diagnostics since the introduction of memiotics in 2008. The connection developed through memory interpretation, patched quotations, analysis pattern icons and notation, software narratology, seminar formulations of diagnostic signs, trace linguistics, and pattern-oriented observability. Close reading, marked messages, and semantic units connected interpretation with particular analysis operations. More recent patterns address meaningful names, shared trace worlds, and the skills needed to read them. My current work brings these developments together with diagnostic residues, the humanities, a philosophy of engineering, and Codesign with large language models. I first outline the semiotic concepts used in this account, with examples from diagnostic practice. I then trace this development and explain the recurring question: how does an artifact become evidence about software structure and behavior? The contribution lies in connecting semiotic distinctions to concrete diagnostic artifacts and analysis patterns. Interpretation proposes a meaning; investigation establishes how far that meaning can be accepted.

Keywords Semiotics; Memiotics; software narratology; trace linguistics; analysis patterns; diagnostic evidence; observability; Codesign; Agemic AI

1 The diagnostic question

A memory dump contains values, structures, addresses, and relationships. A trace contains messages selected and produced by particular parts of a system. A dashboard presents a further selection and transformation of observations. In each case, I have something available for inspection. I still need to establish what it means for the problem under investigation.

This is where semiotics enters my work. I use a sign to understand something else in a particular context. An address can identify an object within one scope and become misleading in another. A familiar message can suggest successful completion even when the software has only accepted a request. A repeated sequence can suggest a known analysis pattern while leaving its cause unresolved.

My interest has therefore developed around a practical relation between artifacts, interpretations, and diagnostic claims. I can recognize a pattern, explain why I recognized it, and ask what additional evidence would distinguish the resulting hypotheses. The history of this relation is distributed across short articles, pattern descriptions, seminar transcripts, books, and more recent manuscripts. Bringing them together makes a line of development visible.

2 A working overview of semiotics

Semiotics studies signs and the processes through which they signify and are interpreted. Semiosis is the process in which something functions as a sign [65, section 2]. I select here the distinctions used throughout this article and illustrate them through my own diagnostic applications.

2.1 Signs and interpretations

In Saussure's account of language, a sign unites a signifier, the sound-image, and a signified, the concept. The sound-image is a psychological impression, and the signified is not the external object [62, Part I, chapter 1]. When adapting this perspective to diagnostic text, I consider the recognizable form of a message and the concept attached to it. The word success can evoke completion even when its documented meaning is only acceptance. The vocabulary needs distinctions that correspond to the states I want to investigate.

Peirce's account relates a sign, its object, and an interpretant: the interpreting effect through which the sign represents its object [64]. The interpretant is not simply another name for the analyst. Consider a recorded transaction message. The message is the sign, the transaction outcome it purports to report is its object, and my understanding that the transaction committed is an interpretant. I still need to check whether the message was produced under conditions that support that understanding. Saussure's concept and Peirce's object therefore have different roles in their respective accounts.

2.2 Icons, indexes, and symbols

Peirce distinguishes signs according to how they represent their objects: through resemblance, an actual connection, or a rule or habit of interpretation [63, sections 3–6]. These distinctions help me examine both recorded artifacts and the representations constructed from them.

Relation	Basis	Diagnostic example and check
Icon	Resemblance or a corresponding arrangement	An analysis pattern icon can preserve a recognizable arrangement. Resemblance supports comparison; it does not establish occurrence in a particular execution.
Index	An actual connection with the object	A captured record can indicate the execution that produced it. I check provenance, identity, and capture conditions before relying on that connection.
Symbol	A convention, rule, or habit	An event name, error code, or diagram label is interpreted through a learned convention. I check the relevant definition, scope, and version.

These functions can coexist in one representation [63, section 5]. A wait graph may use an iconic arrangement, symbolic labels, and connections to recorded waits. I need to ask which relation supports each claim. A diagram can resemble a deadlock without establishing that the represented execution contained one.

2.3 Syntax, semantics, and pragmatics

Morris distinguishes syntactics, the formal relationships among signs; semantics, their relationships to the objects to which they apply; and pragmatics, their relationships to interpreters [65, section 3]. I use these as complementary questions about a diagnostic representation.

Consider a log entry with the fields request=R42 and status=OK. Its syntax includes the field structure and permitted combinations. Its semantics includes what R42 identifies and what OK means under the relevant schema. Its pragmatics concerns how the entry is used: whether an analyst treats it as a reason to retry, investigate further, or close an incident. A parser can accept the entry while its meaning remains ambiguous. Even a correctly understood status may be insufficient for the decision. This is why a well-formed message, a meaningful message, and sufficient diagnostic evidence require different checks.

2.4 Relations within a sign system

A sign also acquires value through its differences from other signs. Saussure distinguishes syntagmatic relations among elements combined in a sequence from associative relations (now usually called paradigmatic) among alternatives or related elements [62, Part II, chapters 4–5]. In my diagnostic applications, I use the paradigmatic perspective for possible substitutions: success, failure, and timeout in a comparable message position. The syntagmatic perspective concerns their arrangement with request, retry, and completion messages. Replacing a status and rearranging an event sequence change different relationships. These distinctions prepare the later discussion of bugtations, Opposition Messages, and narrative units.

Hjelmslev's account relates an expression plane and a content plane through a sign function [66, section 13]. Expression and content are mutually dependent roles in that relation. This prepares the vocabulary of Software Narrative Planes below. In that particular application, I placed trace narratives and program lines of traces on the expression plane, and source code and design on the content plane [13]. The assignment belongs to the relation being analyzed; it does not make source code the content plane of every possible semiotic account.

2.5 Codes, context, and diagnostic inference

Eco's account examines both codes and sign production [67, chapters 1–3]. A semiotic code concerns conventions through which expressions acquire content; it is not synonymous with program source code. For example, a severity label acquires its documented meaning within a logging convention. It may also carry further associations: an analyst can read ERROR as catastrophic even when the application uses it for a recoverable condition. The distinction between denotation and connotation helps me separate the documented classification from these additional associations [67, section 2.3]. I then examine the actual effect on the system.

Lotman's semiosphere places sign processes within a wider semiotic space, with different languages, boundaries, and acts of translation [68]. This provides background for my later analogy of the memiosphere [11]. In diagnostic practice, application developers, database specialists, and operations engineers may read the same records through different vocabularies. Combining their records requires attention to those differences. An identifier can be preserved during collection while its meaning changes between components or teams.

An interpretation can itself become a sign in a report that another analyst reads. For my purposes, following this process means retaining the source artifact, the transformation applied to it, and the assumptions used in the explanation. A normalized message, a pattern name, and an incident summary each preserve selected relationships. I need to know what was omitted before treating the later account as equivalent to the original evidence.

The same discipline applies to absence. If I expect a completion message and cannot find it, I have an observation that needs interpretation. The operation may not have completed; logging may have been disabled; the collector may have lost the message. Forming a possible explanation is an abductive step of the kind named in Thinking-Based Software Diagnostics [15]. Distinguishing these explanations requires evidence about both execution and observation. Semiotics helps me state what a sign is being taken to mean. Diagnostic investigation determines how far that interpretation is warranted.

3 Memiotics and the interpretation of memory

On 18 June 2008, I introduced memiotics, also called memosemiotics, as a branch of memoretics concerned with the interpretation of computer memory, its meaning, signs, and symbols [1]. Memoretics was my proposed study of computer memory, its snapshots, and its evolution. The definition was

subsequently included in Memory Dump Analysis Anthology, Volume 2, and in Theoretical Software Diagnostics [2, 3].

The distinction remains useful. A description of a memory region can specify its bytes, layout, and location. A diagnostic interpretation relates those properties to an object, an operation, a state, or an earlier event. The second activity depends on the first, but it introduces further assumptions. The same bytes may be interpreted as an integer, an address, a string fragment, or part of an obsolete structure.

Consider a memory region that resembles a valid object. Its fields have plausible values and its address occurs in another structure. This resemblance gives me a candidate interpretation. To accept it, I need to consider allocation state, object lifetime, ownership, type information, and the possibility of reuse. A plausible object representation may be a residue of an earlier object. The diagnostic question concerns the relation between the representation and the object at the relevant time.

In Memory Systems Language, published in February 2010, I considered memory analysis through interconnected structures and symbols and proposed a memory language extending a hierarchy of modeling and implementation languages [4]. This extended the earlier definition toward relationships that an analyst can read. A pointer, a containing structure, and a collection of related objects can carry more diagnostic meaning together than an isolated value.

My review of Daniel Chandler's Semiotics: The Basics records another part of this development. The March 2012 review refers back to an earlier review and describes reading the book intermittently from September 2008. It explicitly connects that reading with memiotics and software narratology [5]. The June 2008 definition of memiotics precedes that recorded starting point. I therefore regard the review as evidence of continuing engagement and application, rather than as the starting date of the entire connection.

4 Substitution and a visual vocabulary

In September 2008, I used a different setting to apply semiotic distinctions. Bugtations are modified quotations. In Bugtations: A Semiotic Approach, I described their construction through paradigmatic substitution within a syntagmatic arrangement [6]. The arrangement provides a recognizable form; a replacement changes what that form says. My later description of bugtations as patched quotations connected this activity with the vocabulary of software patching [7].

This was a small literary experiment, but it exposed a useful operation. Meaning changes when I substitute one element while preserving enough of the surrounding structure for the original relation to remain recognizable. Looking back, I can connect that operation with comparisons in diagnostics: an expected message replaced by another message, a familiar stack with one changed frame, or a recurring sequence whose final event differs. These diagnostic comparisons are a retrospective connection, not a claim that the original bugtations article supplied a diagnostic procedure.

The introduction of icons for memory dump analysis patterns in March 2010 added a visual vocabulary [8]. In the first article, I credited Algorithms in a Nutshell with the pattern icon idea and presented representations of Multiple Exceptions (user mode). In my 2025 account, I described the subsequent development of 101 icons, with black-and-white equivalents, over approximately a year and a half [9]. A picture could make a recurring arrangement recognizable across particular debugger displays and platform details.

In October 2010, Notation for Memory and Trace Analysis added a symbolic form to this graphical vocabulary [47]. I proposed capital letters for major pattern categories, such as W for wait chains and D for deadlocks, with subscripts or small letters for subcategories. A slash could combine categories and subcategories. The aim was a concise grammar with readable syntax and semantics. This made the arrangement of signs itself an object of design: a notation should let me express a diagnostic distinction

and let another analyst recover it. The proposal also anticipated notation for pattern succession and spatial differentiation.

There are several relations to keep separate here. A diagram can resemble the structure of a pattern. Its name and conventional elements can depend on learned meanings. A particular diagnostic display can also be connected to the process that produced the observation. These relations can coexist, but a diagram's resemblance alone does not establish that the represented condition occurred in a particular system.

In Memioart, introduced in June 2011, I made the artistic extension explicit. I considered icons and symbols that signify computer memory signs and proposed commentaries explaining their choices [10]. The work crossed between diagnostic representation and visual interpretation. It also made a recurring concern more visible: an image needs a context of use if the viewer is to understand what it represents.

5 From memory environments to software narratives

In March 2011, I introduced the memiosphere by analogy with the semiosphere. I described memory processes as creating, manipulating, and destroying signs [11]. This shifted attention from an isolated memory artifact toward the environment in which such artifacts acquire and lose meaning. The analogy offered a way to think about interacting memory processes; it did not by itself provide a formal model of their behavior.

Commodities as Memories, published in February 2012, extended my interest beyond software diagnostics. Within the proposed perspective of memoidealism, I considered commodities through memories and their connections, in dialogue with accounts centered on signs and value [12]. This belongs to the wider philosophical setting of my work. Its relevance here is the recurrence of memory and interpretation across domains.

The narrative direction had already received a definition in June 2009. In Software Narratology: A Definition, I applied literary narratology to software execution and distinguished four levels [44]. Instructions, statements, and functions corresponded to events; a concrete execution path corresponded to history; the software trace was the narrative; and its display in a trace viewer was the presentation. These levels help explain why changing a trace view need not change the underlying execution, although it can change what an analyst notices.

Two Readings of a Software Trace, published in May 2010, distinguished reconstructing a linear source-code account from constructing an interpretation of reported behavior [45]. The post called the first reading deconstruction, in a sense distinct from the later deconstructive reading discussed below. In the interpretive reading, I removed irrelevant information and compressed activity regions into a trace constructed around the discovered patterns and problem description. Such an account is an analytical construction. Its omissions and rearrangements need to remain accountable to the recorded evidence.

In November 2010, Close and Deconstructive Readings of a Software Trace made the borrowed reading practices explicit [46]. Close reading examined a trace as a unity of messages, looking for structural patterns, repetition, similarity, and contrast. Deconstructive reading examined conflicting or absent messages and the different components contributing to the trace. I read this distinction as an early way of testing whether a coherent diagnostic story conceals a disagreement among its sources.

Implementation Discourse, introduced the following month, connected message structure, syntax, and semantics with the implementation environment [48]. A .NET component, a file system driver, and a COM component can speak differently in their traces. I called the corresponding field Software Trace Linguistics and referred to applying the ethnography of speaking to software narration. Traces of Individuality later added naming and tracing conventions associated with an individual engineer or a corporate coding standard [52]. The interpretation of a message can therefore depend on both the technical environment and the conventions of its producer.

In *Software Narrative Planes*, published on 11 March 2012, I adapted the distinction between expression and content planes from glossematics. I placed the software trace narrative, with its narremes and corresponding program lines of traces, or PLOTs, on the expression plane. Source code and design, with their own construction narremes, belonged to the content plane [13].

The same article distinguished narratives of construction, concerning what ought to be, from narratives of post-construction, concerning what is. Requirements, interactions, software traces, and incident descriptions could then be examined through related but different narrative planes. *Software Narratology Square* combined these planes into a wider arrangement [14].

The practical consequence is that a trace is already an organized expression. Its producers decide which events to describe, what names to use, and which relationships to preserve. An incident report adds another organization. An analyst may then rearrange observations into a proposed explanation. I need to distinguish the order in which events occurred, the order in which messages became available, and the order in which I tell the diagnostic story.

A coherent narrative can help me understand an incident. Its coherence is also something to examine. Missing observations, different viewpoints, and incompatible message meanings may remain hidden inside an apparently continuous account.

My *Introduction to Software Narratology* gives a more concrete account of these distinctions. I explained how selecting and rearranging events produces different plots from a possible full software story. I then introduced paratext and an extended software trace that includes supporting material such as scenario descriptions, pictures, problem histories, and user interviews. Interpretation can therefore require more than the sequence of messages itself [35, pp. 38–40].

The same seminar introduced a software narreme as a basic narrative unit with a semantic aspect. Enter/exit and request/reply were examples; state reports, action reports, and commentary were distinguished as types [35, pp. 41–42]. The earlier February 2012 post *Narremes in Software Narratology* had considered a message invariant and a macromessage serving a semantic function as candidate levels, placing the possible narreme between invariants and macrofunctions [50]. This records an exploration of meaningful units in software narratives. I keep it distinct from the later definition of *Traceme*, whose description uses a different semiotic vocabulary and a particular trace example.

Traces and Logs as Pseudocode develops another reading of software narration [61]. It proposes reconstructing a behavioral account by recognizing control structures and recurring sequences in recorded messages. Within that analogy, analysis patterns supply parts of a grammar for reading execution. Comparing the reconstructed account with intended behavior can expose a difference worth investigating. The paper calls this computational hermeneutics. I treat the pseudocode as an abstraction supported by the available trace, whose coverage may be incomplete; it does not recover every detail of the original program.

6 Diagnostic signs in the seminar transcripts

The seminar *Introduction to Philosophy of Software Diagnostics*, first printed in 2013, provides a direct connection between a diagnostic pattern and a sign. I called such a pattern a *da-sign*: a sign in a dump artifact that concerns an issue I am trying to understand. The name also referred to Heidegger's *Dasein*. The diagnostic use concerned an encountered pattern of software structure or behavior and the understanding attached to it [36, pp. 177–178].

A *da-sign* belonged to a wider interpretive arrangement. I used *prob-lemma* for a tentative problem proposition associated with incomplete understanding. *Da-signs* were meaning-patterns, and *CARE* provided their underlying meaning-structure [36, pp. 173–178]. I explicitly acknowledged borrowing the terms *meaning-pattern* and *meaning-structure* from a work on the phenomenology and hermeneutics of

medicine [36, p. 186]. The distinction placed the reported difficulty, the artifact pattern, and the organization of findings in relation to one another. In this retrospective account, I treat that arrangement as a way to develop and examine an understanding; the presence of a pattern does not by itself guarantee a complete explanation.

The seminar also placed dialogue and interpretation within software diagnostics. A user's account, an engineer's questions, and an execution narrative can contribute different information. Omissions and incomplete interaction records can change the interpretation of a problem. The discussion of hermeneutics connected these narratives with problem description patterns, and the explanation stage returned from identified da-signs to a refined prob-lemma [36, pp. 179–190]. This provides the human context for the sign found in an artifact.

Pattern-Oriented Software Forensics made the vocabulary of signs part of a revised definition of software diagnostics. I described diagnostic problems through indicators or symptoms, defined a forensic pattern through recurring indicators and recommendations in context, and replaced the earlier emphasis on abnormal structure and behavior with the study of signs of software structure and behavior. The transcript also states that forensic and diagnostic patterns share an underlying sign structure and interpretation [37, pp. 132–139]. Pattern-Oriented Memory Forensics repeats this definition and applies the common pattern language across platforms and analysis tools [40, pp. 176–178]. It also describes the reuse of the narratological trace analysis catalogue for memory analysis [40, p. 187].

An explicit mention of semiotics occurs in the closing slide of Pattern-Oriented Software Forensics. The slide carries a 2013 copyright notice and, under the heading What's Next for 2014?, lists Semiotics of Debugging among planned seminars; the transcript was first printed in 2014 [37, p. 162]. This records a named direction in the programme. The slide is an announcement and does not establish that the proposed seminar was subsequently delivered. It nevertheless connects the vocabulary of da-signs and diagnostic indicators with an explicitly semiotic subject.

These seminar formulations add an intermediate stage to the history. Memory was already being interpreted through signs, and software traces were already being read as narratives. The seminars made the pattern itself a sign within an account of a problem and made interpretation part of the diagnostic practice.

7 Semiotic thinking and diagnostic education

Thinking-Based Software Diagnostics places semiotic thinking alongside critical and systemic thinking and connects diagnosis with inductive, deductive, and abductive reasoning [15]. This gives semiotics a role within diagnostic reasoning. I examine how a sign supports an interpretation, how that interpretation participates in an inference, and what conditions the inference requires.

Theoretical Software Diagnostics and Education makes the educational connection explicit by including semiotics among the disciplines relevant to understanding diagnostic signs [3]. Learning a debugger command gives me an operation. Learning an analysis pattern gives me a recognizable arrangement and a vocabulary for discussing it. Learning to examine the sign relation helps me ask when that arrangement supports the proposed conclusion.

The 2012 announcement of Introduction to Philosophy of Software Diagnostics preceded the published seminar treatment discussed above. General Abnormal Patterns of Structure and Behavior proposed further connections between patterns across disciplines [16, 17]. Together, these contributions place semiotic thinking within the wider development of theoretical software diagnostics.

In What I'm passionate about?, published on 1 September 2019, I explicitly listed “Narratology, literary theory, philology, hermeneutics, and semiotics (mostly application to diagnostics and anomaly detection)” among my interests [18]. The same article connected my interest in linguistics with earlier work on C++

semantics and its expansion to human languages and associated theories. I also connected my interests with reading, finding patterns, and making connections. This records the place of semiotics within my wider diagnostic interests between the earlier work on memory and narratives and the later trace analysis patterns.

8 Semiotics within trace analysis patterns

The trace and log analysis pattern language provides a more concrete setting. The fifth edition of *Trace, Log, Text, Narrative, Data* identifies a set of patterns under Narratological and Mathematical Influences, with a discussion that also names semiotics and related fields [19]. This is a combined account of influences. It does not imply that every pattern in the list has an exclusively semiotic origin.

Macrofunction, introduced in November 2011, is an example of meaning formed across messages [49]. I borrowed the distinction between macrofunctions and microfunctions from functionalist linguistics. Opening a connection, sending an SQL command, executing it, and closing the connection can participate in one larger semantic unit. The messages need not belong to the same thread of activity. Recognizing that unit requires relating their roles, rather than treating each line as an independent statement.

Marked Message followed in January 2012 [51]. I explicitly drew on Roman Jakobson's marked and unmarked categories and borrowed distinctive-feature notation from phonology. An analyst could annotate whether a trace showed session initialization, database queries, or a process launch. The original description also preserved an important limitation: a negative feature could mean that an activity was undetected or that it was definitely absent. I must resolve that difference before using a missing sign as evidence that an event did not occur.

Opposition Messages provides an explicit connection. Its December 2012 description connects the name with binary opposition in Saussure's structuralism [42]. Other patterns, including Dialogue, Diegetic Messages, Silent Messages, Trace Viewpoints, and Palimpsest Messages, draw attention to relationships, perspectives, omissions, and layers within diagnostic texts. Their individual definitions remain important. A shared interpretive vocabulary does not make their detection conditions interchangeable.

The earlier seminars also show how interpretation becomes a practical analysis operation. The case study in *Software Trace and Memory Dump Analysis* follows a feature name through registry access messages and combines Vocabulary Index, Bifurcation Point, Data Flow, and Guest Component patterns to explain why a feature becomes available only after another application runs [38, p. 119]. *Mobile Software Diagnostics* uses Paratext for supplementary operating-system information that helps interpret a memory analysis finding [39, p. 280]. These examples concern the use of names and context in an investigation; they were presented as pattern applications.

Pattern-Oriented Network Trace Analysis extends the narrative account to network artifacts. It connects name resolution with a more meaningful presentation, partitions a trace into narrative segments, and discusses paratext as additional interpretive information [41, pp. 268–272, 291, 295]. A representation can make a diagnostic relationship easier to recognize while preserving the need to check what the names and segments denote.

Semantic Field, introduced in July 2020, organizes messages by a category or subject [53]. Its name came from linguistics, and the description allowed fields to be formed through the analysis of Implementation Discourse. I distinguished a Semantic Field, the meaningful category into which messages are grouped, from a Trace Field, the mapping from messages to another domain. This distinction becomes useful when several transformations are involved: the category, the assignment of messages to it, and the presentation of the result are separate parts of the analysis.

In February 2023, I introduced Traceme using an example involving an object address and reference count. The description connects tracing meaning with semes and places the traceme at a higher structural

level than sememes [20]. The point is to identify a meaningful unit that can be followed through a trace. Its significance depends on what the recorded elements refer to and how they relate.

Trace Molecule followed the next day. It connected tracemes into a larger representation and drew inspiration from semic molecules and semic analysis [21]. Here a metaphor helps organize relationships among meaningful trace elements. The resulting representation still needs to be interpreted through the software and tracing context. Its usefulness does not depend on treating the chemical analogy as a literal account of execution.

The Iconic Trace pattern, introduced in March 2023, extended the visual direction [22]. In the later observability article, I related such representations to the syntactic, semantic, and pragmatic content of messages, message blocks, and applied analysis patterns [9]. Syntax concerns arrangement; semantics concerns what the elements mean; pragmatics concerns their use in the diagnostic situation. A transformation of a trace may preserve one of these while losing a distinction needed by another.

For example, grouping identical message text can reveal repetition. If the grouping removes object identity or lifecycle position, it may also combine events that need different interpretations. A trace visualization is therefore part of the diagnostic method. I need to ask which relationships it preserves and which it makes difficult to see.

Semantic Mapping, introduced in June 2026, makes the presentation operation explicit [54]. Opaque identifiers such as process IDs, thread IDs, request IDs, and handles can be mapped to names such as UI Thread, Blocking Thread, or Failed Request. The transformation can also rename schema fields, for example PID to Process. I distinguish this rewriting of labels from grouping messages into a Semantic Field. A useful diagnostic name makes a role visible; its assignment still has to be justified by the evidence.

Trace Ontology extends the analysis from visible vocabulary to entities, classes, relations, and constraints [55]. A trace can contain every expected identifier while leaving its diagnostic role unclear. I need to determine whether a value names a request, an attempt, a session, or an operation, and how those entities relate. Semantic Mapping can give them readable names; Trace Ontology supplies the structured model in which those names are used.

Trace World brings several traces and Implementation Discourses into a shared diagnostic ontology [56]. Application, security, infrastructure, and AI traces may describe connected events using different terms and conventions. The analysis has to preserve identity across those descriptions and examine incompatible meanings or causal stories. I therefore use Trace World for the common diagnostic world reconstructed from those sources. Collecting the traces together is only the beginning of that reconstruction.

Trace Skill adds the competence needed to interpret both the messages and the reconstructed world [57]. Relevant evidence can be present while remaining unrecognized by an analyst who lacks the necessary domain knowledge. Trace Viewpoints concern the perspectives from which a trace is read; Trace Skill concerns the ability to read from them correctly. I read this as returning the pattern language to diagnostic education and collaboration. A shared vocabulary helps specialists exchange findings, while their different skills help establish what the signs mean in each domain.

9 Observability as the production of interpretable signs

The concern with producing interpretable observations was already present in Introduction to Software Narratology. I called it anticipatory software construction: describing software events during construction with later problem analysis in mind. The seminar connected better event descriptions with the needs of maintainers and technical support engineers [35, pp. 47–48]. This gives an earlier formulation of the design concern that I later discuss under observability.

Pattern-Oriented Observability, Part 2, published in June 2025, brought several earlier developments together under Semiotics of Memory, Trace, and Log Analysis [9]. The article revisited memory analysis icons and Iconic Traces and connected them with Dia|gram, Space-like Narratology, and the Lov language. It was also included in Memory Dump Analysis Anthology, Volume 17, pages 67–68 [23].

Dia|gram, introduced in 2016, provides a diagnostic analysis language with visual representations of structure and behavior [9, 24]. Space-like Narratology and the Lov language extend the range of representations considered in the 2025 account [24]. I now read these developments as a movement from recognizing signs in existing artifacts toward deliberately constructing representations for diagnostic use.

The 2025 article used the Peircean distinction between icons and indexes to discuss these representations. In applying that distinction, I need to preserve the difference between resemblance and an actual connection to what is represented. A drawn dependency can resemble a relationship. An observed wait can be connected to a particular execution. The diagnostic interpretation depends on how the representation was produced and what warrants that connection.

Observability makes this an engineering question. If I want to distinguish acceptance from completion, the system must make that distinction observable. If an identifier can be reused, I need enough scope to tell its occurrences apart. If a collector can lose messages, an absence in its output needs to be interpreted with that possibility in mind.

Adding observations changes the available sign system. Event names, schemas, timestamps, retention rules, aggregation, and visualization all affect which interpretations become possible. Their adequacy depends on the diagnostic questions I expect to ask.

Pattern-Oriented Diagnostics and Observability of World Models develops the representational consequence [59]. Traces and logs are already selective models of execution, shaped by instrumentation, schemas, sampling, and storage constraints. A later analytical model operates on that representation. A discontinuity in the recorded narrative may therefore originate in the tracing arrangement as well as in execution. The paper also considers message invariants as semantic anchors: distinctions that help preserve identity and meaning when a representation changes.

Pattern-Oriented Diagnostics and Observability as a Philosophy of Engineering examines interpretive practice [58]. Drawing on early and later Wittgenstein, I relate correspondence and meaning in use to problem patterns and analysis patterns. Problem patterns describe recurring failures; analysis patterns govern how artifacts are selected, transformed, compared, and interpreted. In this account, analysis patterns function as a diagnostic grammar. The question and the analysis operation affect what an artifact can establish.

The same account connects symbolic notation and Dia|gram with rule-following and judgment [58]. A notation expresses pattern combinations; its appropriate use depends on training and context. I see a practical semiotic connection here: diagnostic signs become usable within a practice that specifies how their meanings are examined.

10 Residues and the recent synthesis

My manuscript in preparation, Semiotics of Software Diagnostics, Observability, and Debugging, develops these connections across memory dumps, traces, logs, metrics, debugger output, reports, and artifacts produced by AI systems [25]. It brings together different semiotic traditions while retaining the particular distinctions needed for each diagnostic problem. Its worked investigations use constructed evidence to make the interpretive steps explicit.

My manuscript in preparation, Residues in Diagnostics, provides a complementary emphasis [26]. An artifact is available because something has remained, been recorded, or been reconstructed. A symbolic residue may preserve a value, a name, an identifier, or a relationship. Its diagnostic meaning depends on

its material carrier and on the history of its production and preservation. A residue can survive after the condition that originally gave it meaning has changed.

The historical and comparative studies extend the context. History of Diagnostics, Observability, and Debugging in Humanities includes semiotics and narratology in its account of traces and their organization [27]. Twenty-One Grammars of Diagnostic Evidence returns to the distinction between a sign that is observed and a cause that is inferred [28]. These perspectives help me examine assumptions that become difficult to notice within a single technical vocabulary.

Five Strands of Diagnostics, Observability, and Debugging considers semiotics as a possible unifying perspective [29]. Its strength lies in explaining mediation and meaning: how something becomes evidence for someone in a context. Its limits become apparent when I need to select a discriminating test, justify an intervention, or verify a correction. Those tasks require additional reasoning about mechanisms, uncertainty, consequences, and results.

My more recent Diagnostics, Observability, and Debugging in Sciences, Humanities, and Arts extends the comparison across natural sciences, formal sciences, social sciences, humanities, arts, and engineering sciences. Its humanities discussion places semiotics and narratology among its organizing perspectives and ties terminology, the source of a statement, and the order of a reconstructed narrative to diagnosis [43, pp. 8–9]. The wider synthesis examines relationships among artifacts, representations, recognized patterns, claims, and interventions, including the problems of shared labels and dependent sources [43, pp. 12–16]. I read this as a further application of the semiotic question: which relationship makes an observed sign relevant to the claim? The comparison preserves differences among empirical support, formal validity, interpretive justification, and verification in engineered systems [43, pp. 20–21].

I have also continued to share reading interests around signs, meaning, and language models, including Davide Picca’s Not Minds, but Signs and Paul Kockelman’s Mathematical Models of Meaning, and to provide discussion spaces on LinkedIn and Facebook for semiotics and diagnostic reasoning [30]. These activities belong to the reception and discussion of the subject. They should be distinguished from the diagnostic concepts and methods I have proposed.

11 Codesign and signs of artificial agency

Codesign: Between Code and Design brings the semiotic thread into my current work with large language models [31]. I now spend more attention on design decisions and on assessing proposed implementations. Generated code makes a possible design available for examination. It may also make an omitted condition or a misunderstanding visible.

One reading of Codesign is CODE + SIGN. A function, a type, or a group of modules can act as a sign of a design decision. The code also executes. I therefore examine both the represented decision and the mechanism that implements it. A name such as ReliableClient invites an interpretation; its retry behavior and failure handling determine whether it meets the requirement.

The other readings connect the overlap of CODE and DESIGN with co-designing through interaction. CODE + SIGNER adds responsibility for accepting an implementation for a specified use. I use these readings as a mnemonic combination, while recognizing the established meanings and history of co-design.

This returns me to the diagnostic relation. A generated explanation, a success message, a passing test, and an observed external effect are different artifacts. Each supports a different range of claims. My responsibility includes deciding what evidence is sufficient for the decision I am making.

Agemic AI vs. Agentic AI extends the concern with meaningful distinctions to artificial agency [60]. By analogy with phonemics, I propose an ageme as a minimal functionally contrastive unit of agency. Candidate units include goal commitment, retrieval, monitoring, repair, and termination. Their status

depends on whether a change in the unit changes the organization of agency, rather than merely changing an implementation detail. The article explicitly connects the term with the emic tradition in anthropology and semiotics.

An ageme, a narreme, and a traceme address different questions. The proposed ageme concerns a contrast in the organization of agency; the narreme concerns a narrative unit; the traceme concerns trace meaning [20, 50, 60]. One tool invocation may participate in all three descriptions, but the occurrence of that invocation does not establish their equivalence. For diagnostics, the useful question is which distinction makes a missing, substituted, or unsuccessful action recognizable in the evidence.

12 Following a sign through an investigation

Consider a constructed example. A service is intended to receive a request, write a record, and return an acknowledgement. The client reports a timeout. A log entry contains the word success. A later retry appears to produce a duplicate effect. The example is deliberately small so that the interpretive steps remain visible.

I first ask what success names. It may mean that the request was accepted, that a write was attempted, that a transaction committed, or that an acknowledgement was sent. The text alone does not settle the matter. I need the event's production rule and its relation to the operation.

I then examine the identifiers. A request identifier, an attempt identifier, and a business-operation identifier may refer to different objects. If I treat them as interchangeable, two attempts can appear to be two independent operations, or two operations can appear to be one. Their scope is part of their meaning. Trace Ontology makes these roles and relationships explicit, and Semantic Mapping can give the relevant values readable names [54, 55]. If I combine client, server, and database traces, Trace World requires me to establish which entities are shared across their different discourses [56].

Next I examine the narrative. The client's timeout describes its observation. It does not establish that the server failed to commit. The order in which the collector received messages may also differ from the order of the underlying events. A narrative that moves directly from timeout to failure can therefore conceal the possibility of a committed operation whose acknowledgement was lost.

The residues can distinguish the hypotheses. A durable commit record, an idempotency record, an attempt history, and an external observation of the resulting effect have different roles. I would examine their provenance, retention, and relation to the same operation. Marked Message reminds me to distinguish an acknowledgement that was not detected from an acknowledgement known not to have occurred [51]. Trace Skill also matters: interpreting a database commit may require knowledge that the client-side investigator does not possess [57]. A familiar pattern can direct this search, but the resulting evidence determines which explanation survives.

Finally, I can turn the investigation into an observability requirement. I may need separate events for acceptance, commitment, and acknowledgement, together with identifiers that preserve operation and attempt scope. I also need to know whether the collection path can omit those events. This is the engineering consequence of the semiotic analysis: preserve the distinctions on which the diagnostic decision depends.

13 Contribution and limits

Computer semiotics has an established history. Andersen's Computer Semiotics presents computer systems as targets of interpretation; de Souza's semiotic engineering examines communication between designers and users; Tanaka-Ishii's Semiotics of Programming considers programming through a semiotic perspective [32–34]. My work belongs within this wider setting.

The contribution I trace here is more specific. Memiotics named a focus on the interpretation of computer memory. Pattern icons, symbolic notation, and later visual languages gave recurring diagnostic arrangements representations that could be recognized and discussed. Software narratology, trace reading, and trace linguistics connected diagnostic texts with semantic units, discourse, and interpretive operations. The seminar formulations connected patterns with signs, their meaning-structures, and diagnostic interaction. Semantic Mapping, Trace Ontology, Trace World, and Trace Skill extended the account to naming, shared representations, and competence. Pattern-oriented observability and the philosophy of engineering addressed how evidence is produced and used. The recent studies, manuscripts, and Codesign connect these developments with residues, AI-produced artifacts, and responsibility for interpretation. Agemic AI proposes a further application to the organization of artificial agency.

This is a claim about the development and integration of my work. It is not a claim to have introduced semiotics into computing, nor does the chronology establish priority over every related approach. The early definitions, individual patterns, retrospective interpretations, and current proposals have different evidential status.

The practical value of a semiotic distinction depends on what it helps me do. It may expose an ambiguity, preserve a reference, reveal a lost relationship, or identify an unsupported transition from observation to explanation. To establish an improvement in diagnostic performance, I would still need suitable comparisons and evidence. A more expressive vocabulary is useful when it makes reasoning more inspectable and decisions better supported.

The recurring question remains concrete. What does this artifact allow me to say about this software, under these conditions? Memiotics began with the interpretation of memory. The same question now reaches traces, logs, observability, diagnostic reports, and executable proposals produced during Codesign.

Chronology of the semiotic thread

The sequence below separates the dates of early proposals from later accounts that connect and extend them. Manuscripts in preparation and undated web publications are grouped separately at the end.

Period	Development	Connection
2008	Memiotics and bugtations	Memory interpretation and substitution within recognizable forms [1, 6].
2009	Software narratology	Execution events, history, trace narrative, and presentation distinguished [44].
2010–2011	Signs, reading, and trace language	Memory language, icons, symbolic notation, trace readings, Implementation Discourse, Macrofunction, Memiosphere, and Memioart [4, 8, 10, 11, 45–49].
2012	Narrative units and linguistic distinctions	Marked Message, narremes, narrative planes, anticipatory construction, and Opposition Messages [5, 12–14, 16, 35, 42, 50, 51].
2013–2016	Patterns, education, and representation	Da-signs, meaning-patterns, sign-based diagnostics, Traces of Individuality, education, and Dia gram [3, 17, 24, 36, 37, 40, 52].
Planned for 2014	Semiotics of Debugging	Named in a closing slide © 2013, printed in 2014; the announcement alone does not establish delivery [37, p. 162].
2019	Semiotics among my stated interests	Explicit connection with diagnostics and anomaly detection in What I’m passionate about? [18].
2020	Semantic classification	Semantic Field groups messages by subject or category, borrowing from linguistics [53].
2023	Meaningful trace structures	Traceme, Trace Molecule, and Iconic Trace [20–22].
2025	Semiotics of observability	Explicit retrospective connection of memory, trace, and log representations [9, 23].
June–July 2026	Names, shared worlds, and skills	Semantic Mapping, Trace Ontology, Trace World, and Trace Skill connect presentation, entities, discourse, and competence [54–57].
2026	Historical synthesis and Codesign	Historical, comparative, and design-oriented studies [27–29, 31, 43].
In preparation	Semiotics and diagnostic residues	A sustained semiotic account and the interpretation of diagnostic residues [25, 26].
Undated web publications	Interpretation and further proposals	Engineering philosophy, trace world models, Agemic AI, and traces as pseudocode [58–61].

Seminar source passages

The page references below distinguish the position in each combined PDF from the page number printed within its individual seminar. Software Diagnostics refers to SoftwareDiagnostics-CollectedSeminars-Revision2.pdf; Principles refers to PrinciplesMemoryDumpAnalysis-Revision2.pdf. The cited seminar texts include revisions dated 2015. The named semiotics item is a planned topic; the other rows identify developed discussions of signs, interpretation, or narrative representation.

Collection and seminar	PDF / seminar pages	Relevant passage
Software Diagnostics — Software Narratology [35]	38–42 / 15–19	Plot selection, paratext, extended traces, and semantic narrative units.
Software Diagnostics — Software Narratology [35]	47–48 / 24–25	Anticipatory software construction and event descriptions for later analysis.
Software Diagnostics — Philosophy of Software Diagnostics [36]	173–178 / 11–16	Prob-lemma, da-sign, meaning-pattern, and CARE as meaning-structure.
Software Diagnostics — Philosophy of Software Diagnostics [36]	179–190 / 17–28	Dialogue, interpretation, hermeneutics, explanation, and terminological borrowing (PDF p. 186).
Principles — Pattern-Oriented Software Forensics [37]	132–139 / 13–20	Indicators, signs, forensic patterns, and the revised diagnostic definition.
Principles — Pattern-Oriented Software Forensics [37]	162 / 43	Explicit planned seminar title: Semiotics of Debugging.
Principles — Software Trace and Memory Dump Analysis [38]	119 / 25	Trace case study connecting a feature name with a diagnosis and workaround.
Principles — Mobile Software Diagnostics [39]	280 / 22	Paratext supporting interpretation of a Spiking Thread finding.
Principles — Pattern-Oriented Memory Forensics [40]	176–178 / 14–16; 187 / 25	Signs in execution artifacts, a common pattern language, and narratological reuse.
Software Diagnostics — Pattern-Oriented Network Trace Analysis [41]	268–272 / 17–21; 291 / 40; 295 / 44	Narrative structure, name resolution, partitioning, and paratext.

References

Web sources accessed 12, 16, and 17 October 2026. Version labels identify the copies cited. Section titles are used where pagination varies between editions and exports. References 35–41 identify the seminar transcripts. Their page citations in the article use the combined PDF positions; the source map gives corresponding printed seminar pages.

- [1] Vostokov, D. (2008). Memiotics: A Definition. 18 June. [Article](#)
- [2] Vostokov, D. Memory Dump Analysis Anthology, Volume 2. “Memiotics,” p. 350. OpenTask. See the collected anthology contents. [Contents](#)
- [3] Vostokov, D. (2024). Theoretical Software Diagnostics: Collected Articles. Fourth edition. OpenTask. “Memiotics,” p. 55; “Thinking-Based Software Diagnostics,” p. 175; “Theoretical Software Diagnostics and Education,” p. 255. ISBN 9781912636914.
- [4] Vostokov, D. (2010). Memory Systems Language, Part 1. 12 February. [Article](#)
- [5] Vostokov, D. (2012). Semiotics: The Basics. Book review, 29 March. Links to Chandler’s second edition (Routledge, 2007; ISBN 0415363756); records reading from September 2008 and a first review written in 2008. [Review](#)
- [6] Vostokov, D. (2008). Bugtations: A Semiotic Approach. 16 September. [Article](#)
- [7] Vostokov, D. (2010). Bugtations as Patched Quotations, Bugtation No. 115. 3 March. [Article](#)
- [8] Vostokov, D. (2010). Icons for Memory Dump Analysis Patterns, Part 1. 8 March. [Article](#)
- [9] Vostokov, D. (2025). Pattern-Oriented Observability, Part 2: Semiotics of Memory, Trace, and Log Analysis. LinkedIn publication dated 6 June; also available at DumpAnalysis.org. [Article](#) · [Dated publication](#)
- [10] Vostokov, D. (2011). Memioart: The New Art Form. 22 June. [Article](#)
- [11] Vostokov, D. (2011). Memiosphere. 9 March. [Article](#)
- [12] Vostokov, D. (2012). Commodities as Memories, Part 1. 21 February. [Article](#)
- [13] Vostokov, D. (2012). Software Narrative Planes. 11 March. [Article](#)
- [14] Vostokov, D. (2012). Software Narratology Square. 11 March. [Article](#)
- [15] Vostokov, D. (n.d.). Thinking-Based Software Diagnostics. Collected in Memory Dump Analysis Anthology, Vol. 7, p. 434 (articles written November 2011–May 2014), and Theoretical Software Diagnostics, fourth edition, p. 175. [Article](#)
- [16] Vostokov, D. (2012). Webinar: Introduction to Philosophy of Software Diagnostics. Announcement, 23 September, for a webinar scheduled for 17 December. [Announcement](#)
- [17] Vostokov, D. (2013). General Abnormal Patterns of Structure and Behavior, Part 0. 29 January. [Article](#)
- [18] Vostokov, D. (2019). What I’m passionate about? LinkedIn article, 1 September. [Article](#)
- [19] Vostokov, D. (2023). Trace, Log, Text, Narrative, Data: An Analysis Pattern Reference for Information Mining, Diagnostics, Anomaly Detection. Fifth edition, revision 5.06, August 2026. OpenTask. “Narratological and Mathematical Influences” and the individual pattern entries. ISBN 9781912636587.
- [20] Vostokov, D. (2023). Trace Analysis Patterns, Part 219: Traceme. 6 February. [Pattern](#)
- [21] Vostokov, D. (2023). Trace Analysis Patterns, Part 220: Trace Molecules. Listed as Trace Molecule in the fifth-edition pattern reference. 7 February. [Pattern](#)
- [22] Vostokov, D. (2023). Trace Analysis Patterns, Part 230: Iconic Trace. 6 March. [Pattern](#)
- [23] Vostokov, D. (2025). Memory Dump Analysis Anthology, Volume 17. “Semiotics of Memory, Trace, and Log Analysis,” pp. 67–68 in the online PDF edition. OpenTask. ISBN 9781912636174. [Publication page](#)
- [24] Vostokov, D. Dia|gram: Diagnostic Analysis Language; Space-like Narratology; and Lov Language. Dia|gram dates to 2016 [9] and is collected in Memory Dump Analysis Anthology, Vol. 9b, p. 82 (articles March–September 2016). The later representations are discussed in reference 9. [Dia|gram](#) · [Space-like Narratology](#) · [Lov](#)
- [25] Vostokov, D. Semiotics of Software Diagnostics, Observability, and Debugging. Manuscript in preparation. Especially chapters 8, 25, 29, and 31–37.

- [26] Vostokov, D. Residues in Diagnostics. Manuscript in preparation. Section 8.2 on symbolic residues.
- [27] Vostokov, D. (2026). History of Diagnostics, Observability, and Debugging in Humanities. Version 3. Chapter on semiotics, narratology, and the organization of traces. [Publication page](#) · [PDF](#)
- [28] Vostokov, D. (2026). Twenty-One Grammars of Diagnostic Evidence. Version 15. Discussions of Sanskrit and Greek, pp. 75 and 116 in the PDF. [Publication page](#) · [PDF](#)
- [29] Vostokov, D. (2026). Five Strands of Diagnostics, Observability, and Debugging: A Technological and Intellectual Synthesis. Version 20, September. Semiotics and candidate unifying perspectives, pp. 25 and 27–28. [Publication page](#) · [Version 20 PDF](#)
- [30] Vostokov, D. LinkedIn posts sharing Davide Picca’s Not Minds, but Signs: Reframing LLMs through Semiotics (2025, arXiv:2505.17080) and Paul Kockelman’s Mathematical Models of Meaning: A Dynamic Systems Approach to Possible World Semiotics (MIT Press, 2025; ISBN 9780262552684). Related discussion groups on LinkedIn and Facebook. [LLM post](#) · [Meaning post](#) · [LinkedIn group](#) · [Facebook group](#) · [Picca paper](#) · [Kockelman book](#)
- [31] Vostokov, D. (2026). Codesign: Between Code and Design. Codesign with LLMs and the Software Codesigner Profession. Version 9, 8 October. [Publication page](#) · [PDF](#)
- [32] Andersen, P. B. (1992). Computer Semiotics. Scandinavian Journal of Information Systems, 4(1), Article 1. [Journal record](#)
- [33] de Souza, C. S. (2005). The Semiotic Engineering of Human-Computer Interaction. MIT Press. ISBN 9780262042208. [Publisher](#)
- [34] Tanaka-Ishii, K. (2010). Semiotics of Programming. Cambridge University Press. ISBN 9780521736275. [University book record](#)
- [35] Vostokov, D. (2012). Introduction to Software Narratology. Revision 2.0, June 2015. OpenTask. ISBN 9781908043078. In Software Diagnostics, supplied Revision 2 collection, PDF pp. 24–50; seminar pp. 1–27.
- [36] Vostokov, D. (2013). Introduction to Philosophy of Software Diagnostics. Revision 2, July 2015. OpenTask. ISBN 9781908043573. In Software Diagnostics, supplied Revision 2 collection, PDF pp. 163–197; seminar pp. 1–35.
- [37] Vostokov, D. (2014). Pattern-Oriented Software Forensics. First printing, 2014; Revision 2, July 2015. OpenTask. ISBN 9781908043696. In Principles of Memory Dump Analysis, supplied Revision 2 collection, PDF pp. 120–162; seminar pp. 1–43. The closing slides carry a 2013 copyright notice and preserve plans for 2014; the year in this reference is the first-printing year.
- [38] Vostokov, D. (2014). Software Trace and Memory Dump Analysis. Lecture text © 2011; first printing, 2014; Revision 2.0, May 2015. OpenTask. ISBN 9781908043238. In Principles of Memory Dump Analysis, supplied Revision 2 collection, PDF pp. 95–119; seminar pp. 1–25. The cited case study is an addition to the lecture transcript and is not dated here to the original lecture.
- [39] Vostokov, D. (2013). Mobile Software Diagnostics. Revision 2.0, July 2015. OpenTask. ISBN 9781908043658. In Principles of Memory Dump Analysis, supplied Revision 2 collection, PDF pp. 259–283; seminar pp. 1–25.
- [40] Vostokov, D. (2014). Pattern-Oriented Memory Forensics. Revision 2, July 2015. OpenTask. ISBN 9781908043764. In Principles of Memory Dump Analysis, supplied Revision 2 collection, PDF pp. 163–258; seminar pp. 1–96.
- [41] Vostokov, D. (2013). Pattern-Oriented Network Trace Analysis. Revision 2, July 2015. OpenTask. ISBN 9781908043580. In Software Diagnostics, supplied Revision 2 collection, PDF pp. 252–302; seminar pp. 1–51. Cited passages: pp. 268–272, 291, and 295 of the combined PDF.
- [42] Vostokov, D. (2012). Trace Analysis Patterns, Part 62: Opposition Messages. 14 December. [Pattern](#)
- [43] Vostokov, D. (2026). Diagnostics, Observability, and Debugging in Sciences, Humanities, and Arts: An Overview and Synthesis of Six Histories. Version 5, October. Sections 5, 9–12, 14, and 15; pp. 8–9, 12–16, and 20–21. [Publication page](#) · [PDF](#)
- [44] Vostokov, D. (2009). Software Narratology: A Definition. 12 June. [Article](#)
- [45] Vostokov, D. (2010). Two Readings of a Software Trace. 21 May. [Article](#)
- [46] Vostokov, D. (2010). Close and Deconstructive Readings of a Software Trace. 3 November. [Article](#)

- [47] Vostokov, D. (2010). Notation for Memory and Trace Analysis, Part 1. 7 October. [Article](#)
- [48] Vostokov, D. (2010). Trace Analysis Patterns, Part 34: Implementation Discourse. 7 December. [Pattern](#)
- [49] Vostokov, D. (2011). Trace Analysis Patterns, Part 44: Macrofunction. 22 November. [Pattern](#)
- [50] Vostokov, D. (2012). Narremes in Software Narratology. 2 February. [Article](#)
- [51] Vostokov, D. (2012). Trace Analysis Patterns, Part 45: Marked Message. 2 January. [Pattern](#)
- [52] Vostokov, D. (2013). Trace Analysis Patterns, Part 65: Traces of Individuality. 14 March. [Pattern](#)
- [53] Vostokov, D. (2020). Trace Analysis Patterns, Part 195: Semantic Field. 31 July. [Pattern](#)
- [54] Vostokov, D. (2026). Trace Analysis Patterns, Part 260: Semantic Mapping. 27 June. [Pattern](#)
- [55] Vostokov, D. (2026). Trace Analysis Patterns, Part 263: Trace Ontology. 6 July. [Pattern](#)
- [56] Vostokov, D. (2026). Trace Analysis Patterns, Part 264: Trace World. 8 July. [Pattern](#)
- [57] Vostokov, D. (2026). Trace Analysis Patterns, Part 265: Trace Skill. 10 July. [Pattern](#)
- [58] Vostokov, D. (n.d.). Pattern-Oriented Diagnostics and Observability as a Philosophy of Engineering. [Publication page](#) · [PDF](#)
- [59] Vostokov, D. (n.d.). Pattern-Oriented Diagnostics and Observability of World Models: A Topological Perspective. [Publication page](#) · [PDF](#)
- [60] Vostokov, D. (n.d.). Agemic AI vs. Agentic AI: Toward an Agemics of Artificial Agency. [Article](#)
- [61] Vostokov, D. (n.d.). Traces and Logs as Pseudocode. Especially sections 3–6 and 9. [PDF](#)
- [62] Saussure, F. de (2011 [1916]). Course in General Linguistics. Translated by Wade Baskin; edited by Perry Meisel and Haun Saussy. Columbia University Press. Part I, chapter 1; Part II, chapters 4–5. ISBN 9780231157278. [Publisher](#)
- [63] Peirce, C. S. (1894). What Is a Sign? In The Essential Peirce, Volume 2: Selected Philosophical Writings (1893–1913), edited by the Peirce Edition Project. Indiana University Press, 1998, pp. 4–10. Especially sections 2–6. [Text](#) · [Publisher](#)
- [64] Peirce, C. S. (1909). Manuscript 634, pp. 23–28. September. Transcription by Joseph Ransdell, presented as “Charles Peirce on the object of a sign as cause of the sign.” Arisbe. [Transcription](#)
- [65] Morris, C. W. (1938). Foundations of the Theory of Signs. International Encyclopedia of Unified Science, Vol. 1, No. 2. University of Chicago Press. Especially sections 2–3, pp. 3–9. [Repository record and PDF](#)
- [66] Hjelmslev, L. (1961 [1943]). Prolegomena to a Theory of Language. Translated by Francis J. Whitfield. Revised English edition. University of Wisconsin Press. Section 13, “Expression and Content.” [Scholarly bibliography](#)
- [67] Eco, U. (1976). A Theory of Semiotics. Indiana University Press. Chapters 1–3, especially sections 1.2, 2.2–2.3, and 3.1. [Publisher](#)
- [68] Lotman, J. (2005 [1984]). On the Semiosphere. Translated by Wilma Clark. Sign Systems Studies, 33(1), 205–229. DOI 10.12697/SSS.2005.33.1.09. [Article](#)