Draft version January 20, 2026
Typeset using LaTeX **twocolumn** style in AASTeX7.0.1

Specification–Implementation Balance: A Diagnostic Framework for AI-Mediated Software Construction

Egor Merkushev[1]

[1] *Independent Researcher*

ABSTRACT

We introduce a non-normative diagnostic framework for observing the balance between specification (intent), implementation (code), and AI inference cost in modern AI-assisted software development. Rather than proposing optimization targets or productivity metrics, the framework is designed to expose structural and economic shifts in the process of intent-to-implementation conversion.

Keywords: software engineering — specification-driven development — AI-assisted programming — software metrics

INTRODUCTION

The increasing use of large language models in software development has blurred the boundary between specification and implementation. Code is no longer written exclusively by humans, while specifications increasingly act as executable or semi-executable artifacts. This shift challenges traditional code-centric metrics such as LOC, velocity, or test coverage.

In this work, we propose a set of observability metrics that characterize *where semantic intent resides* and *how it is transformed into executable systems.*

1. ANALYTIC FRAMEWORK

1.1. *Specification as Intent Atoms*

We represent specification as a set of discrete *Intent Atoms*, denoted N_{spec}, composed of: user stories, acceptance criteria, invariants, and architectural decisions.

1.2. *Implementation as Functional Output*

Implementation is represented by two orthogonal quantities: (i) the observable functional output N_{func}, and (ii) the implementation size S_{impl}, measured in lines of code, tokens, or normalized AST units.

1.3. *AI Inference Footprint*

We define the inference footprint as the tuple $(C_{\text{inf}}, T_{\text{inf}}, N_{\text{calls}})$, corresponding to monetary cost, token volume, and non-tokenizable operations (e.g., external tool invocations, search queries, or discrete API actions).

2. DERIVED METRICS

2.1. *Specification–Implementation Balance*

We define the Specification–Implementation Balance (SIB) as

$$\text{SIB} \equiv \frac{N_{\text{spec}}}{S_{\text{impl}}}. \tag{1}$$

This quantity characterizes the distribution of semantic intent between explicit specification and executable realization. No optimal value is assumed.

2.2. *Intent Yield*

The Intent Yield (IY) is defined as

$$\text{IY} \equiv \frac{N_{\text{func}}}{N_{\text{spec}}}. \tag{2}$$

IY captures how much observable functionality is produced per unit intent.

2.3. *Intent Conversion Cost*

The Intent Conversion Cost (ICC) is defined as

$$\text{ICC} \equiv \frac{C_{\text{inf}}}{N_{\text{spec}}}. \tag{3}$$

This metric reflects the economic cost of AI-mediated intent realization.

2.4. *Functional Cost*

The Functional Cost (FC) is given by

$$\text{FC} \equiv \frac{C_{\text{inf}}}{N_{\text{func}}}. \tag{4}$$

2.5. *Operational Yield*

The Operational Yield (OY) is defined as

$$\mathrm{OY} \equiv \frac{N_{\mathrm{func}}}{N_{\mathrm{calls}}}. \tag{5}$$

OY captures how much observable functionality is produced per discrete non-tokenizable operation. It characterizes the efficiency of intent-to-implementation conversion through the lens of operational effort.

3. DISCUSSION

These metrics are explicitly non-normative. They are not intended for optimization, ranking, or performance evaluation. Instead, they provide a structured lens for analyzing semantic allocation and economic mediation in software systems.

4. CONCLUSION

We argue that AI-assisted software development requires a shift from code-centric metrics toward intent-centric observability. Specification–Implementation Balance offers a compact formalism for reasoning about this transition.

5. RELATION TO EXISTING METRICS

The metrics introduced in this work are not intended to replace existing software engineering metrics. Instead, they operate at a different abstraction level, focusing on the semantic and economic mediation between specification and implementation. In this section, we clarify the relation between our framework and several widely used metric families.

5.1. *Function Points*

Function Points (FP) measure externally observable functionality from the perspective of a system user, independent of implementation details. They are commonly used for cost estimation and productivity analysis.

In our framework, Functional Units (N_{func}) play a role analogous to Function Points, but with two important differences. First, Functional Units are treated as a *proxy* for observable behavior rather than as a standardized accounting unit. Second, we explicitly avoid interpreting N_{func} as a productivity measure.

The Functional Cost metric,

$$\mathrm{FC} \equiv \frac{C_{\mathrm{inf}}}{N_{\mathrm{func}}}, \tag{6}$$

can be viewed as an AI-era analogue of FP-based cost models. However, unlike classical FP analysis, FC is not intended to support estimation or optimization, but rather to expose the economic footprint of AI-mediated realization.

5.2. *Requirements Traceability*

Requirements traceability aims to establish explicit links between requirements, design artifacts, code, and tests. Traceability matrices and coverage ratios are commonly used to ensure completeness and compliance, particularly in safety-critical systems.

Our framework does not attempt to enforce or reconstruct explicit trace links. Instead, it introduces aggregate observables that reflect the *distribution of intent* across artifacts. The Specification–Implementation Balance,

$$\mathrm{SIB} \equiv \frac{N_{\mathrm{spec}}}{S_{\mathrm{impl}}}, \tag{7}$$

can be interpreted as a coarse-grained signal of traceability drift: significant changes in SIB over time may indicate that intent is migrating toward implicit representation in code or, conversely, being externalized into specifications.

Thus, while traceability answers the question *"Is this requirement implemented?"*, SIB addresses the orthogonal question *"Where does the system's meaning primarily reside?"*

5.3. *Test and Coverage Metrics*

Code coverage metrics (e.g., line, branch, or path coverage) quantify the extent to which implementation is exercised by tests. In behavior-driven or test-driven development, tests themselves often act as executable specifications.

The Intent Yield metric,

$$\mathrm{IY} \equiv \frac{N_{\mathrm{func}}}{N_{\mathrm{spec}}}, \tag{8}$$

is complementary to coverage metrics. Coverage measures how much of the implementation is validated, whereas Intent Yield measures how effectively declared intent materializes as observable behavior.

Importantly, a system may exhibit high test coverage while still showing low Intent Yield, indicating overspecification, architectural fragmentation, or excessive decomposition of intent. Conversely, a high Intent Yield with low coverage may signal fragile or under-validated functionality.

5.4. *Summary of Conceptual Differences*

Existing metrics predominantly operate within a single artifact domain (code, tests, or requirements). The framework proposed here is explicitly cross-artifact and non-normative, emphasizing relationships rather than absolute quantities.

Rather than optimizing for higher or lower values, the proposed metrics are intended to support longitudinal

observation and qualitative reasoning about system evolution in AI-assisted development contexts.

6. THREATS TO VALIDITY

As this work proposes a conceptual and diagnostic framework rather than an empirical evaluation, several limitations and threats to validity must be explicitly acknowledged.

6.1. *Construct Validity*

The proposed metrics rely on abstract quantities such as Intent Atoms (N_{spec}) and Functional Units (N_{func}), which are necessarily proxy representations of semantic intent and observable behavior. Different teams or projects may adopt divergent decompositions of specifications into stories, criteria, or decisions, leading to variability in absolute values.

To mitigate this threat, the framework explicitly avoids normative interpretation of metric values and focuses on relative changes (Δ-based analysis) within a single project over time.

6.2. *Measurement Subjectivity*

The identification and counting of Functional Units may be influenced by architectural style, domain granularity, or interface conventions. For example, a single coarse-grained API endpoint and a collection of fine-grained endpoints may represent comparable semantic functionality while yielding different N_{func}.

This framework therefore treats Functional Units as a qualitative observable rather than a standardized accounting measure, analogous to the use of architectural metrics in early-stage system design.

6.3. *Dependence on AI Tooling*

Inference-related metrics such as Intent Conversion Cost and Functional Cost depend on external AI model providers, pricing schemes, and prompting strategies. Changes in model efficiency, pricing, or context window size may alter absolute values without reflecting any intrinsic change in system structure or intent.

For this reason, inference-related quantities are intended to be interpreted within fixed tooling configurations or reported in coarse-grained cost bands rather than as precise numerical targets.

6.4. *External Validity*

The framework is primarily motivated by AI-assisted and specification-driven development workflows. Its applicability to traditional, fully human-authored development processes may be limited, particularly in contexts where specifications are informal or absent.

However, the framework does not assume a specific programming language, model architecture, or development methodology, suggesting that its core concepts may generalize across a wide range of software systems.

6.5. *Risk of Metric Misuse*

As with any quantitative metric, there exists a risk of misuse if the proposed quantities are reinterpreted as performance indicators or optimization objectives. Such use would directly contradict the diagnostic intent of the framework and may lead to behavior consistent with Goodhart's law.

We therefore emphasize that the proposed metrics are intended solely for observability and interpretive analysis, not for ranking, benchmarking, or incentive alignment.