Decision Analysis

> Effect-Oriented Structural Scope Audit for AI-Assisted Software Development

Author: Spark Tsai
ORCID: https://orcid.org/0009-0006-8847-4703
Email: spark.tsai@gmail.com
Date: March 2026

Abstract

AI-assisted code generation introduces a structural gap between what an AI system is authorized to modify and what it actually modifies. Testing validates outcome correctness but not scope authorization. Code review evaluates quality but not structural boundary compliance. Monitoring tracks operational metrics but not decision attribution. None of these mechanisms answer a prior question: did the generation stay within its authorized scope?

This paper introduces Decision Analysis, an effect-oriented analytical framework that determines whether observable effects of AI-assisted code generation fall within structurally authorized boundaries. The framework inherits its structural vocabulary from three prior works: Anchor Architecture provides the coordinate system $(\mathcal{A}, R)$, Viewpoint-Structured Specification provides decision carriers as specification elements (V, E), and Boundary provides the execution-time admissibility resolution (V', E') and $B(t)$.

Decision Analysis introduces two constructs not present in any upstream work. First, **Decision Effect**, defined as an observable state deviation $E_t = \Delta(S_{t_0}, S_t)$, serves as the analytical entry point. Second, **three Anchor-based Scopes** — Visible Scope S_v, Artifact Scope S_a, and Outcome Scope S_o — decompose the execution lifecycle into pre-authorized context, intended modification target, and actual modification result.

Five mutually exclusive analytical states — **DA-N (Normal), DA-O (Over Outcome), DA-I (Indeterminate), DA-V (Decision Vacancy), DA-U (Unobservable)** — are defined as set-relational conditions over S_v, S_a, and S_o. These states support a pre-merge structural scope audit: an automated determination of whether each generation event remained within its authorized boundary, without requiring semantic understanding of the generated code.

The framework provides review triage (directing human attention to scope-exceeding changes), cumulative scope tracking (detecting cross-session boundary erosion), and constraint quality feedback (using state distributions as a basis for governance adjustment). Risk interpretation is explicitly excluded; governance implications are addressed in a companion Decision Risk paper.

Keywords

Decision Analysis, Decision Effect, Structural Scope Audit, AI-Assisted Software Development, Anchor Architecture, Pre-Merge Analysis, Scope Authorization, Effect-First Analysis, Code Generation Governance

1. Introduction

AI-assisted code generation has created a structural condition for which no existing engineering practice provides adequate instrumentation. The condition is not defective code, insufficient testing, or inadequate review. It is the absence of any mechanism that determines whether the output of a generation event falls within the scope it was authorized to modify.

1.1 Four Engineering Failures

Four observable failures motivate this work. Each arises from the interaction between stateless AI generation and stateful system evolution, and none is addressable by existing outcome-oriented validation practices.

Cumulative boundary erosion. Each AI generation session is stateless. A Boundary established for the first session does not persist into the third. By the fifth session, the AI system has no awareness of the original scope authorization. Even if each individual session produces modifications within its locally determined scope, the cumulative modification set may far exceed the original Boundary. This is the structural source of Inference Creep [Tsai, 2026b]: not a single act of boundary violation, but a series of locally reasonable expansions whose aggregate exceeds any declared authorization.

Review throughput asymmetry. A single generation event can produce hundreds of lines of code in seconds. A development session may involve dozens of such events. Human review capacity does not scale proportionally. When production rate exceeds review rate, review necessarily becomes

sampling. Sampling misses scope-exceeding modifications that are functionally correct — the modifications that are hardest to detect because they produce no test failures and no obvious defects.

Constraint governance without feedback. Constraints — prohibitive rules that bound the permissible action space of a generation event — require periodic adjustment as systems evolve. Without a quantitative feedback mechanism, constraint governance operates without a Check phase: constraints are set but never systematically evaluated for adequacy. A constraint that was appropriate at system version n may be insufficient at version $n+k$, but no existing mechanism provides evidence of this insufficiency.

Merge without scope verification. Code produced by AI generation enters the codebase through the same merge process as human-authored code. Pull request review evaluates whether the code is correct, well-structured, and consistent with project conventions. It does not evaluate whether the modifications correspond to the scope authorized by the originating directive. A functionally correct modification to a file outside the authorized scope passes review and enters the codebase as a latent structural anomaly.

1.2 Why Existing Practices Fail

These four failures share a common structural cause: existing validation practices are outcome-oriented, not scope-oriented.

Testing answers: *Is the behavior correct?* Testing validates that outputs match expected behavior. A modification that changes session timeout from 30 minutes to 24 hours passes all authentication tests if no test explicitly checks session duration. Testing does not ask whether the session timeout was within the scope authorized for modification.

Code review answers: *Is the code good?* Review evaluates quality, consistency, and correctness. A reviewer examining an authentication-related pull request focuses on the authentication logic. A concurrent modification to session management configuration, included in the same diff, may receive insufficient attention — not because the reviewer is negligent, but because the review process provides no structural signal that this modification falls outside the authorized scope.

Monitoring answers: *Is the system healthy?* Operational monitoring tracks metrics, error rates, and performance indicators. It detects deviations from operational baselines. It does not attribute those deviations to specific generation events, and it cannot determine whether a detected deviation originated from an authorized or unauthorized modification.

The question that none of these practices answers is:

> Did this generation event modify only what it was authorized to modify?

This is the question Decision Analysis is designed to answer.

1.3 Contributions

This paper makes three contributions:

1. **Decision Effect as an analytical primitive.** Effect — defined as observable state deviation — is extracted from its implicit distribution across testing, monitoring, and review, and established as a unified, first-class analytical entry point. The Effect-First Principle is argued to be the only structurally feasible analysis starting point in AI-assisted environments where intent is not introspectable and outcome correctness is insufficient.
2. **Three Scopes as Anchor Sets.** Visible Scope S_v (pre-authorized context from specification), Artifact Scope S_a (intended modification target), and Outcome Scope S_o (actual modification result) decompose the generation lifecycle into structurally comparable Anchor Sets. The five analytical states are defined as set-relational conditions over these three Scopes.
3. **Pre-merge structural scope audit.** A diagnostic protocol enables automated determination of analytical states prior to code merge, supporting review triage, cumulative scope tracking, and constraint quality feedback within a PDCA governance cycle.

2. Structural Foundations

Decision Analysis inherits its structural vocabulary from three prior works. This section declares the specific constructs inherited from each, establishes a term mapping, and states the structural precondition for analysis.

Decision Analysis introduces no new structural primitives beyond Decision Effect (Section 3) and three Scopes (Section 3.4). All other structural concepts are used as defined in their source works.

2.1 From Anchor Architecture: Coordinate System

Anchor Architecture [Tsai, 2026e] defines two primitives:

- **Anchor** $\mathcal{A} = \langle ID, L, \tau \rangle$: a spatiotemporal coordinate binding identity to a persistent location and temporal index.
- **Relationship** $R \subseteq \mathcal{A} \times \mathcal{A}$: a directed binary relation over the anchor set.

From these primitives, the following constructs are available to Decision Analysis:

- **Transitive closure** R^+: reachability over declared relations.
- **resolve()**: content resolution via coordinate.
- **trace()**: path enumeration over R^+ from source to target.
- **impact()**: forward reachability set from a given anchor.

Decision Analysis uses $(\mathcal{A}, R)$ as the coordinate system within which all three Scopes are defined and all analytical operations are expressed. No modification or extension of Anchor Architecture is required.

2.2 From VSS: Decision Carriers and Visible Scope

Viewpoint-Structured Specification [Tsai, 2026] defines:

- **Specification** as a structured artifact $\mathrm{Specification} = (V, E)$, where V is a set of declared Viewpoints and E is a set of persistently addressable specification elements.
- **Three sufficiency conditions** for each element $e \in E$: Persistent Addressability, Explicit Scope, and Viewpoint Membership.

Decision Analysis uses VSS elements as **Decision Carriers** — artifacts capable of hosting or propagating a decision. A carrier in this framework is a specification element $e \in E$ that satisfies all three sufficiency conditions and is therefore persistently addressable, explicitly scoped, and anchored to a declared Viewpoint.

The Visible Scope S_v (defined in Section 3.4) originates from the human selection of (V', E') — the subset of the Specification chosen as the governing context for a specific execution event.

2.3 From Boundary: Artifact Scope and Constraint

Boundary as an Execution-Time Primitive [Tsai, 2026f] defines:

- **Task Execution Boundary** $B(t) = \{a \mid a \in Possible(S_t) \wedge \forall c \in C, \neg c(a)\}$: the resolved permissible action space at execution time.
- **Visible Scope** S_t: the set of artifacts accessible to the model during execution.
- **Prohibitive Constraints** C: explicit MUST NOT predicates defining invariants.
- **Boundary** as selection from Specification: (V', E') where $V' \subseteq V$, $E' \subseteq E$.

Decision Analysis uses the Boundary framework to define Artifact Scope S_a (Section 3.4) — the intended modification target, determined either by direct human specification or by AI derivation from S_v through R.

Constraint granularity — whether constraints operate at directory, file, function, or behavioral level — affects the precision of S_a and consequently the analytical resolution of Decision Analysis. However, Decision Analysis does not evaluate constraint adequacy; it observes the relationship between actual outcomes and declared scopes regardless of how those scopes were established.

2.4 Term Mapping

The following table establishes the correspondence between Decision Analysis concepts and their source definitions. Decision Analysis does not redefine any concept that already has a formal definition in an upstream work.

DA Concept	Source	Formal Origin
Carrier	VSS	Specification element $e \in E$ satisfying sufficiency conditions
Trace	AA	Path in R^+
Outcome Boundary	Boundary	(V', E') + admissibility predicate $B(t)$
Visible Scope S_v	VSS → Boundary	Anchor Set derived from (V', E') selection
Artifact Scope S_a	Boundary	Anchor Set of intended modification targets
Outcome Scope S_o	**DA (new)**	Anchor Set of actual modifications
Effect E_t	**DA (new)**	Observable state deviation

2.5 Structural Precondition

Decision Analysis requires the existence of all three upstream structures:

$$\text{DA executable} \iff \exists\,(\mathcal{A}, R) \wedge \exists\,(V, E) \wedge \exists\, B(t)$$

The absence of any single foundation collapses analysis:

Missing Foundation	Consequence
Anchor Architecture	No coordinate system; trace operations undefined; analysis collapses to DA-U
VSS	No carrier structure; S_v undefined; boundary check impossible
Boundary	No S_a definition; "scope-exceeding" is undecidable; DA-O indistinguishable from DA-N

This three-source dependency is a feature, not a limitation. It ensures that Decision Analysis operates on structurally grounded evidence rather than interpretive inference.

3. Decision Effect

3.1 The Implicit Status of Effect in Current Practice

The concept of effect — a detectable change in system state resulting from an action — is present in every engineering validation practice. Yet no existing practice treats effect as a first-class analytical object with its own definition, observability condition, and analysis protocol.

In testing, effect is subsumed by outcome. A test asserts that a specific output matches an expected value. The underlying state deviation that produced the output is not an independent object of analysis. A test that passes confirms outcome correctness; it provides no information about whether the state deviation that produced the outcome was within authorized scope. Testing observes a projection of effect, not effect itself.

In monitoring, effect is subsumed by metrics. An operational dashboard displays latency, error rates, and throughput. A change in any metric signals a state deviation, but the metric provides no attribution — it does not connect the deviation to a specific generation event, a specific carrier, or a specific scope authorization. Monitoring observes a symptom of effect, not effect itself.

In code review, effect is subsumed by diff. A reviewer examines the textual difference between two code versions. The diff represents a structural modification, which is one category of effect. But review treats the diff as an object to be evaluated for quality, not as an object to be evaluated for scope authorization. Review applies judgment to effect, but does not analyze effect as a structural object.

The result is that effect is distributed across three practices, each of which observes a different projection of the same underlying phenomenon, and none of which treats the phenomenon itself as an analytically independent entity.

Decision Analysis extracts effect from these projections and establishes it as a unified analytical primitive.

3.2 Formal Definition: Effect as State Deviation

A **Decision Effect** is a detectable state deviation:

$$E_t = \Delta(S_{t_0}, S_t)$$

Where:

- S_{t_0} is the system state prior to a generation event
- S_t is the system state after a generation event
- Δ is the state-difference operator

An effect may manifest as:

- Source code modification
- Configuration change
- Interface exposure or modification
- Dependency addition or removal
- Structural reorganization
- File creation or deletion

Effect is strictly broader than output. An output is what a generation event produces as its declared result. An effect is any detectable state deviation, including modifications that the generation event did not declare as its purpose.

Effect is strictly narrower than operational deviation. An operational deviation (latency increase, error rate change) may result from causes unrelated to any generation event. Effect, as defined here, is anchored to a generation event through temporal correspondence: E_t is observed within the temporal scope of a specific execution.

3.3 Observability as Entry Condition

Effect observability is binary:

$$\mathcal{O}(E_t) \in \{0, 1\}$$

$\mathcal{O}(E_t) = 1$ if and only if the state deviation is detectable through the available anchoring infrastructure. Detection does not require semantic understanding of the deviation; it requires only that the deviation is structurally visible — that is, that the modified artifacts are anchored in $\mathcal{A}$ and their pre- and post-states are resolvable.

$\mathcal{O}(E_t) = 0$ if no state deviation is detectable. This may occur because:

- No generation event occurred
- The generation event produced no modifications
- Modifications occurred but were not anchored (the artifacts were not in $\mathcal{A}$)

Observability is the entry condition for all subsequent analysis. When $\mathcal{O}(E_t) = 0$, analysis terminates at DA-U (Section 4.5). No further determination is possible or attempted.

This binary condition eliminates the ambiguous state of "possible but uncertain effect." An effect is either structurally visible or it is not. There is no intermediate analytical category.

3.4 Three Scopes within Boundary

Within the Boundary framework, three distinct Anchor Sets decompose the generation lifecycle:

Visible Scope $S_v \subseteq \mathcal{A}$

The set of specification-layer anchors selected as governing context for a generation event. S_v originates from the human selection of (V', E') from the versioned Specification. It represents what the AI system is authorized to see — the governing constraints, requirements, and architectural specifications that define the intent for this execution.

S_v is determined **before execution** by human decision.

Artifact Scope $S_a \subseteq \mathcal{A}$

The set of code-layer anchors identified as intended modification targets. S_a represents what the AI system is expected to modify.

S_a may be established through two mechanisms:

1. **Direct human specification.** The Directive explicitly identifies the files, functions, or modules to be modified (e.g., "modify only `src/auth/` "). S_a is a declared Anchor Set.
2. **AI derivation through R.** The AI system traverses R from specification anchors in S_v to find structurally related code anchors. For example, specification element SRS-042 may be related

through R to functions FUNC-login and FUNC-oauth; the AI determines S_a based on this structural traversal.

In either case, S_a is determined **before or at the initiation of execution**, not after.

When S_a is AI-derived, the derivation itself is subject to analysis: if the AI traverses R^+ beyond a reasonable depth, S_a may already exceed the reachable scope of S_v, producing an analytical anomaly before any code is generated.

Outcome Scope $S_o \subseteq \mathcal{A}$

The set of anchors whose content was actually modified during execution. S_o represents what the AI system actually did — the factual record of modification.

S_o is determined **after execution** by observation. It is not planned, declared, or authorized. It is the structural fact against which authorization is evaluated.

The relationship between S_o and S_a depends on three factors:

- Whether the Directive included Constraints (MUST NOT predicates)
- Whether the AI system complied with those Constraints
- The granularity of the Constraints (directory-level, file-level, function-level, behavioral-level)

Summary:

Scope	Content	Determined by	Timing
S_v	What you can see	Human selection from VSS	Before execution
S_a	What you intend to modify	Human specification or AI derivation	Before / at initiation
S_o	What was actually modified	AI execution (observed)	After execution

All three are Anchor Sets within $(\mathcal{A}, R)$. Their inter-set relations are what Decision Analysis determines.

3.5 Effect-First Principle

Decision Analysis proceeds from observable effect. Analysis does not begin with intent assumptions, outcome evaluation, or authority claims.

$$\text{Observe } S_o \longrightarrow \text{Compare } S_o \text{ vs } S_a \longrightarrow \text{Trace via } R^+ \text{ to } S_v \longrightarrow \text{Determine State}$$

This ordering is not a methodological preference. It is a structural necessity imposed by three constraints of AI-assisted environments:

Intent-first analysis is structurally infeasible. In human-authored development, intent can be recovered through authorship memory: the developer who wrote the code can explain the decision. In AI-assisted development, the generation process is not introspectable. The internal reasoning that led to a specific modification is not available for analysis. Attempting to begin analysis from intent requires information that does not exist in recoverable form. This is the condition defined as Ghost Intent [Tsai, 2026g]: executable artifacts whose originating decision cannot be recovered.

Outcome-first analysis is insufficient. Outcome validation (testing) confirms that the system behaves correctly after modification. But correct outcomes do not imply authorized scope. A modification that changes session timeout from 30 minutes to 24 hours may produce no test failures while constituting a scope violation. Outcome correctness is orthogonal to scope authorization; one cannot be inferred from the other.

Effect-first analysis is the only approach that requires no unavailable information. S_o is observable: it is the structural fact of what was modified. S_a is declared or derivable: it is the structural record of what was intended to be modified. The comparison S_o vs S_a requires no semantic understanding, no intent reconstruction, and no outcome evaluation. It is a set-membership query over Anchor Sets — decidable, automatable, and independent of the complexity of the generated code.

3.6 Effect Is Not Risk

Decision Analysis determines analytical states. It does not determine governance implications.

A state of DA-O (Over Outcome) means that the generation event modified anchors outside its Artifact Scope. It does not mean that the modification is harmful, unauthorized, or in need of remediation. Whether DA-O constitutes a governance risk is a question for Decision Risk [DR], not for this framework.

This separation is structural, not rhetorical. Decision Analysis and Decision Risk serve different analytical roles:

	Decision Analysis	Decision Risk
Input	S_o, S_a, S_v	DA analytical states

	Decision Analysis	Decision Risk
Output	Analytical state ∈ {DA-N, DA-O, DA-I, DA-V, DA-U}	Governance risk classification
Method	Set-relational comparison	Governance interpretation
Evaluative stance	None (structural determination)	Risk evaluation

The separation ensures that analytical determination remains repeatable and governance-neutral. The same S_o, S_a, S_v always produce the same DA state regardless of organizational context, risk appetite, or governance policy.

4. Analytical State Definitions

Five mutually exclusive states are defined over the three Scopes. Each state is a set-relational condition. The conditions are evaluated in a fixed diagnostic order (Section 5.1).

4.1 DA-N — Normal

Condition:

$$S_o \neq \emptyset \ \wedge \ S_o \subseteq S_a \ \wedge \ S_a \subseteq R^+(S_v)$$

The generation event produced observable modifications ($S_o \neq \emptyset$). All modifications fall within the Artifact Scope ($S_o \subseteq S_a$). The Artifact Scope is structurally reachable from the Visible Scope ($S_a \subseteq R^+(S_v)$).

No analytical anomaly. The generation event stayed within its authorized scope at both the artifact and specification levels.

4.2 DA-O — Over Outcome

Condition:

$$S_o \neq \emptyset \ \wedge \ S_o \not\subseteq S_a$$

The generation event produced observable modifications that extend beyond the Artifact Scope. At least one modified anchor is not in S_a.

Two sub-conditions exist:

DA-O within specification reach:

$$(S_o \setminus S_a) \subseteq R^+(S_v)$$

The scope-exceeding modifications are structurally reachable from the Visible Scope. The AI modified artifacts outside the intended target but within the broader specification context. The generation "did more than asked" but the additional work is structurally related to the governing specification.

DA-O beyond specification reach:

$$(S_o \setminus S_a) \not\subseteq R^+(S_v)$$

The scope-exceeding modifications are not reachable from the Visible Scope. The AI modified artifacts that have no structural relation to the governing specification. This is a more severe analytical finding, though severity assessment belongs to the governance layer.

DA-O does not evaluate whether the scope-exceeding modifications are correct, useful, or harmful. It identifies the structural fact of scope exceedance and, through R^+, determines whether the exceedance is specification-related or specification-unrelated.

Possible causal sources include:

- Inference expansion beyond explicit instruction [Inference Creep]
- Constraint absence (no MUST NOT predicates in Directive)
- Constraint non-compliance (AI ignored declared Constraints)
- Constraint granularity insufficiency (directory-level Constraint failed to prevent function-level exceedance)

4.3 DA-I — Indeterminate

Condition:

$$S_o \neq \emptyset \ \wedge \ S_o \not\subseteq S_a \ \wedge \ |trace^{-1}(S_o \setminus S_a) \cap S_v| > 1 \ \wedge \ \nexists\, \ell^*$$

Where:

- $trace^{-1}(S_o \setminus S_a) \cap S_v$ is the set of specification-layer anchors that could plausibly source the scope-exceeding modifications
- ℓ^* is a uniquely attributable source locus

The generation event produced scope-exceeding modifications, and those modifications can be traced back to the specification layer, but the trace yields multiple plausible sources with no stable unique attribution.

This state arises characteristically in multi-Viewpoint environments. When the Directive references both a functional requirement (SRS-042: "add OAuth support") and an architectural constraint (CAS-017: "prohibit circular dependencies"), a structural refactoring in the generated code may be attributable to either source. If $trace^{-1}$ from the refactoring anchor reaches both SRS-042 and CAS-017 through R^{+}, and no structural evidence distinguishes which was the actual driver, the state is Indeterminate.

DA-I does not resolve attribution. It identifies the structural fact of attribution instability.

4.4 DA-V — Decision Vacancy

Condition:

$$S_o \neq \emptyset \ \wedge \ S_o \not\subseteq S_a \ \wedge \ trace^{-1}(S_o \setminus S_a) \cap \mathcal{A}_{known} = \emptyset$$

Where $\mathcal{A}_{known}$ is the entire set of anchors known to the system — specification elements, code anchors, test anchors, configuration anchors.

The generation event produced scope-exceeding modifications, and those modifications cannot be traced to any known anchor. No specification element, no code artifact, no test case, no configuration record provides a plausible source for the modification. The effect exists, but the decision that produced it does not exist in any recoverable structural form.

This is the formal detection criterion for Ghost Intent [Tsai, 2026g]: executable artifacts whose originating decision cannot be recovered.

DA-V is the most analytically severe non-U state. It indicates not merely scope exceedance (DA-O) or attribution ambiguity (DA-I), but complete absence of structural origin.

4.5 DA-U — Unobservable

Condition:

$$S_o = \emptyset$$

No observable modification was detected. Analysis terminates at the entry condition.

DA-U may result from:

- No generation event occurring
- The generation event producing no modifications
- Modifications occurring outside the anchored artifact space (unanchored modifications are structurally invisible)

DA-U is an analytical result, not a risk classification. Whether unobservability constitutes a governance concern is a question for Decision Risk, not for this framework.

4.6 Mutual Exclusivity and Completeness

The five states partition all possible analytical situations:

Entry gate: $\mathcal{O}(E_t)$

$$\mathcal{O}(E_t) = 0 \implies \text{DA-U}$$

$$\mathcal{O}(E_t) = 1 \implies S_o \neq \emptyset \implies \text{proceed to scope comparison}$$

Scope comparison: S_o vs S_a

$$S_o \subseteq S_a \ \wedge \ S_a \subseteq R^{+}(S_v) \implies \text{DA-N}$$

$$S_o \not\subseteq S_a \implies \text{proceed to attribution analysis}$$

Attribution analysis: $trace^{-1}(S_o \setminus S_a)$

$$|trace^{-1}| = 1 \ (\text{unique}) \implies \text{DA-O}$$

$$|trace^{-1}| > 1 \ (\text{multiple, no unique}) \implies \text{DA-I}$$

$$|trace^{-1}| = 0 \ (\text{none}) \implies \text{DA-V}$$

The five states are exhaustive: every combination of observability, scope membership, and attribution is covered. They are mutually exclusive: no analytical situation satisfies the conditions of more than one state.

4.7 Cumulative Analysis

The preceding definitions apply to a single generation event. In practice, AI-assisted development involves sequences of generation events across multiple sessions.

Let $S_o^{(1)}, S_o^{(2)}, \ldots, S_o^{(n)}$ denote the Outcome Scopes of n successive generation events, and let $S_v^{(0)}$ denote the original Visible Scope at the start of the development sequence.

The **cumulative Outcome Scope** is:

$$\Sigma S_o = \bigcup_{i=1}^{n} S_o^{(i)}$$

A cumulative analytical state is determined by comparing ΣS_o against $S_v^{(0)}$:

$$\Sigma S_o \subseteq R^+(S_v^{(0)}) \implies \text{cumulative DA-N}$$

$$\Sigma S_o \not\subseteq R^+(S_v^{(0)}) \implies \text{cumulative DA-O}$$

This captures the condition in which every individual session is DA-N (each $S_o^{(i)} \subseteq S_a^{(i)}$), yet the aggregate modification set exceeds the original Boundary. Cumulative DA-O is the structural detection of Inference Creep: scope expansion that is invisible at the session level but observable at the sequence level.

5. Diagnostic Protocol

5.1 Single-Session Protocol

Given a completed generation event, the diagnostic protocol proceeds:

```
Step 1: Observe
  Determine S_o (actual modifications)
  If S_o = ∅ → DA-U. Terminate.

Step 2: Compare Scope
  Compare S_o against S_a (intended modification target)
  If S_o ⊆ S_a and S_a ⊆ R⁺(S_v) → DA-N. Terminate.

Step 3: Identify Exceedance
  Compute S_o \ S_a (anchors modified outside intended scope)

Step 4: Trace Attribution
  For each anchor in S_o \ S_a:
    Compute trace⁻¹ through R⁺ toward S_v and A_known

Step 5: Determine State
  If unique attribution exists → DA-O
  If multiple attributions, no unique → DA-I
  If no attribution in A_known → DA-V
```

Each step is a set operation or relation traversal over $(\mathcal{A}, R)$. No semantic interpretation is required. The protocol is deterministic: the same S_v, S_a, S_o always produce the same state.

5.2 Cumulative Protocol

Given a sequence of n generation events:

```
Step 1: Accumulate
  Compute ΣS_o = ∪ S_o^(i) for i = 1..n

Step 2: Compare against Original Boundary
  Compare ΣS_o against R⁺(S_v^(0))
  If ΣS_o ⊆ R⁺(S_v^(0)) → Cumulative DA-N
  If ΣS_o ⊄ R⁺(S_v^(0)) → Cumulative DA-O

Step 3: Identify Erosion Points
  Compute ΣS_o \ R⁺(S_v^(0))
  For each anchor: identify which session introduced it
```

The cumulative protocol detects scope erosion that is invisible to per-session analysis. It answers: over the course of this development sequence, has the aggregate modification set remained within the

originally authorized boundary?

5.3 Worked Scenario

Consider an AI-assisted development sequence involving an authentication module.

Setup:

The Specification contains:

- SRS-042: "System shall support OAuth 2.0 authentication" (specification anchor in S_v)
- CAS-017: "Authentication module shall not directly reference session management" (constraint anchor in S_v)

The Directive specifies: "Implement OAuth support in `src/auth/`."

The Artifact Scope S_a contains:

- FUNC-login (`src/auth/login.py`)
- FUNC-oauth (`src/auth/oauth.py`)
- FUNC-validate (`src/auth/validate.py`)

Session 1: DA-N

AI modifies FUNC-login and FUNC-oauth to add OAuth support.

$S_o^{(1)} = \{\text{FUNC-login}, \text{FUNC-oauth}\}$

$S_o^{(1)} \subseteq S_a$. State: **DA-N**.

Session 2: DA-O

AI modifies FUNC-validate and additionally modifies CONFIG-session (`src/session/config.py`) to extend session timeout for OAuth tokens.

$S_o^{(2)} = \{\text{FUNC-validate}, \text{CONFIG-session}\}$

CONFIG-session $\notin S_a$. State: **DA-O**.

$trace^{-1}(\text{CONFIG-session})$ reaches SRS-042 through R^+ (OAuth tokens require session management). The exceedance is specification-related (DA-O within specification reach), but it violates CAS-017's constraint against direct session management references.

Session 3: DA-V

AI adds a utility function FUNC-cache (`src/auth/cache.py`) that implements response caching for OAuth token validation.

$S_o^{(3)} = \{\text{FUNC-cache}\}$

FUNC-cache $\notin S_a.\, trace^{-1}(\text{FUNC-cache}) \cap \mathcal{A}_{known} = \emptyset$. No specification element mentions caching. No requirement, constraint, or design artifact provides a structural source for this function.

State: **DA-V**. This is Ghost Intent — an executable artifact whose originating decision cannot be recovered.

Cumulative Analysis:

$\Sigma S_o = \{\text{FUNC-login}, \text{FUNC-oauth}, \text{FUNC-validate}, \text{CONFIG-session}, \text{FUNC-cache}\}$

$\Sigma S_o \not\subseteq R^+(S_v^{(0)})$ (CONFIG-session and FUNC-cache are outside specification reach).

Cumulative state: **DA-O**, even though Session 1 was individually DA-N.

6. Engineering Integration

6.1 Pre-Merge Scope Audit

Decision Analysis integrates into the CI/CD pipeline as a pre-merge check. After each generation event and before merge approval:

DA State	Label	Action
DA-N	`scope-verified`	Low-priority review; fast-track eligible
DA-O	`scope-exceeded`	Must-review; report includes: exceeding anchors, trace paths, S_v boundary
DA-I	`attribution-ambiguous`	Must-review; report includes: multiple specification sources, request for reviewer attribution
DA-V	`no-origin-found`	Investigation required; report includes: unattributable anchors, Ghost Intent warning

DA State	Label	Action
DA-U	`analysis-incomplete`	Anchoring gap; report includes: unanchored artifacts, request for anchor supplementation

This triage mechanism does not replace code review. It directs reviewer attention to the modifications most likely to contain structural anomalies, reducing the probability that scope-exceeding changes pass review due to throughput constraints.

6.2 Review Triage

When production rate exceeds review capacity, DA states provide a principled basis for allocating review effort:

DA-N generations have passed structural scope verification. Review can focus on code quality and correctness rather than scope authorization. These generations are candidates for expedited review or, under appropriate governance policies, automated approval.

DA-O generations contain modifications outside the intended scope. Review must evaluate whether the scope exceedance is acceptable (intentional expansion) or unacceptable (unauthorized drift). The DA report identifies which specific anchors exceed scope and what specification elements they trace to (if any), providing the reviewer with a structured basis for judgment rather than requiring full-diff inspection.

DA-V generations contain modifications with no structural origin. These require investigation beyond standard code review: the question is not whether the code is good, but why it exists at all.

6.3 PDCA Feedback on Constraints

The distribution of DA states across a development period provides quantitative feedback on constraint adequacy:

Observation	Diagnosis	Recommended Adjustment
DA-O frequent	Constraint granularity insufficient	Refine Constraints from directory-level to file- or function-level
DA-V frequent	Specification coverage insufficient	Extend VSS to cover domains where AI generates without specification basis

Observation	Diagnosis	Recommended Adjustment
DA-I frequent	Specification conflict unresolved	Invoke Semantic Conflict Detection (VSS Section 6.2) to resolve multi-source ambiguity
DA-N dominant	Constraint governance adequate	Maintain current Constraint configuration
DA-U frequent	Anchoring infrastructure incomplete	Extend anchoring coverage to include unanchored artifact domains

This feedback cycle integrates Decision Analysis into the PDCA governance model:

- **Plan:** Select S_v from VSS; set Constraints and S_a in Directive
- **Do:** Execute AI generation under Boundary
- **Check:** Run Decision Analysis; observe DA state distribution
- **Act:** Adjust Constraints, extend Specification coverage, or refine anchoring based on DA feedback

Without the Check phase provided by Decision Analysis, constraint governance operates as Plan-Do-Plan-Do — constraints are set and reset without evidence of their effectiveness.

6.4 Taxonomy Coverage

Decision Analysis provides pre-merge detection capability for a subset of degradation phenomena defined in the Taxonomy of Degradation Phenomena in AI-Generated Code [Tsai, 2026h]:

Phenomenon	DA Detection	Mechanism
Ghost Intent	DA-V	$trace^{-1} = \emptyset$
Inference Creep	DA-O (cumulative)	$\Sigma S_o \not\subseteq R^+(S_v^{(0)})$
Scope Absorption	DA-O (single)	$S_o \not\subseteq S_a$
Semantic Expansion	DA-O	Carrier reinterpretation detected via boundary exceedance
Intent Fragmentation	DA-I	Multiple attribution without stable unique source

Phenomenon	DA Detection	Mechanism
Ghost Code	DA-V	Structurally isolated anchor with no specification origin
Shadow Coupling	DA-O	Modification touches undeclared dependency anchors

Phenomena not detectable by Decision Analysis:

Phenomenon	Reason
Conditional Blindspot	Requires runtime execution; not observable in static scope comparison
State Assumption Leak	Requires runtime state sequencing; not observable pre-merge
Non-Deterministic Degradation	Requires load-dependent observation
Confidence Propagation	Requires runtime behavioral analysis
Observation Bias	Requires monitoring infrastructure analysis

Decision Analysis covers Intent-layer and Structural-layer phenomena. Behavioral-layer phenomena require runtime observation and are outside the scope of pre-merge analysis.

7. Downstream Interfaces

7.1 Decision Analysis → Decision Risk

Decision Risk [DR] interprets DA states as governance risk conditions. The mapping is:

DA State	DR Mapping	Governance Interpretation
DA-N	—	No analytical anomaly; no governance risk produced
DA-O	DR-O	Scope exceedance with attribution; governance can identify the structural source and evaluate authorization

DA State	DR Mapping	Governance Interpretation
DA-I	DR-A	Attribution instability; governance cannot assign responsibility to a unique source
DA-V	DR-V	No structural origin; governance cannot determine whether the decision was legitimate
DA-U	(determined by DR)	Unobservability; whether this constitutes governance risk is a governance-layer question, not an analytical one

Decision Analysis provides this mapping as an interface specification. The governance interpretation of each state — including whether DA-U should produce a DR classification — is the responsibility of the Decision Risk framework.

7.2 Decision Analysis → Observed Phenomena

Decision Analysis provides formal detection criteria for phenomena defined in companion works:

Phenomenon	DA Projection	Reference
Ghost Intent	DA-V under intent-to-code projection	Tsai, 2026g
Inference Creep	Causal source of DA-O (cumulative)	Tsai, 2026b
Semantic Expansion	Carrier reinterpretation producing DA-O	Tsai, 2026 (Taxonomy)

These projections are diagnostic, not definitional. The phenomena are defined in their respective works; Decision Analysis provides the structural conditions under which they can be detected through scope analysis.

8. Discussion

8.1 Necessity of Three-Source Convergence

Decision Analysis cannot operate on fewer than three upstream foundations. This is not an implementation constraint but an analytical one.

Without Anchor Architecture, the three Scopes cannot be expressed as Anchor Sets, and trace operations over R^+ are undefined. The comparison S_o vs S_a degenerates into file-path matching — still useful as a lightweight approximation, but structurally impoverished (it cannot detect function-level exceedance within a file, nor can it trace attribution through transitive relations).

Without VSS, the concept of Visible Scope has no structural basis. S_v becomes "whatever context was provided," which may be an ad-hoc collection of files rather than a structured selection from a versioned Specification. The distinction between "authorized to see" and "happened to be in context" collapses.

Without Boundary, the concept of Artifact Scope has no formal definition. S_a becomes "whatever the AI decided to modify," which conflates intended scope with actual scope — the very distinction Decision Analysis exists to maintain.

The three-source requirement reflects a genuine analytical dependency, not architectural overengineering.

8.2 Lightweight Approximation

The full Decision Analysis framework requires anchoring infrastructure that may not exist in all development environments. A lightweight approximation operates at the file-path level:

- S_v ≈ files referenced in the prompt or specification document
- S_a ≈ files explicitly named in the directive
- S_o ≈ files appearing in the generation diff

The comparison $S_o \subseteq S_a$ at the file level provides a coarse-grained scope check that captures the most obvious exceedances (modifications to files outside the declared scope). It does not support function-level analysis, transitive attribution, or cumulative tracking, but it requires no infrastructure beyond the development tool's existing diff output.

This approximation represents the minimum viable implementation of the Decision Analysis concept. Tools that currently have no scope-checking capability can adopt it immediately; the full framework provides the target architecture for progressive refinement.

8.3 Artifact Scope Derivation and Pre-Execution DA-O

When S_a is derived by the AI system through R^+ rather than directly specified by a human, the derivation itself may produce scope exceedance. If the AI traverses R^+ to an unreasonable depth, S_a may include anchors that are structurally distant from S_v, producing a condition where $S_a \not\subseteq R^+(S_v)$ at a reasonable traversal depth even before any code is generated.

This implies that Decision Analysis may identify DA-O at two stages:

1. **Pre-execution:** $S_a \not\subseteq R^+(S_v)$ at bounded depth (the AI's own scope determination exceeds the specification reach)
2. **Post-execution:** $S_o \not\subseteq S_a$ (the actual modifications exceed the AI's own scope determination)

Both are instances of scope exceedance, but they have different causal structures. Pre-execution DA-O suggests a problem with scope derivation; post-execution DA-O suggests a problem with generation discipline.

8.4 DA-U as Analytical Boundary

DA-U is not a failure state. It is the boundary of the analytical domain.

When $S_o = \emptyset$, Decision Analysis has no input to analyze. This may reflect a genuine absence of modification or a gap in anchoring coverage. In either case, the appropriate response is not to infer risk but to acknowledge the limit of analytical capability.

The temptation to treat DA-U as a risk indicator ("we can't see what happened, so something bad might have happened") belongs to the governance layer. Decision Analysis reports the absence of observable effect and makes no further claim.

8.5 Exclusions

This work explicitly excludes:

- **Governance interpretation.** Whether a DA state constitutes risk, requires intervention, or triggers policy action is addressed in Decision Risk.
- **Accountability allocation.** Decision Analysis identifies structural attribution (which carrier is the source); it does not determine human responsibility.
- **Code quality evaluation.** Decision Analysis evaluates scope authorization, not correctness, efficiency, or maintainability.
- **Runtime behavioral analysis.** Behavioral-layer phenomena (Conditional Blindspot, State Assumption Leak, Non-Deterministic Degradation) require runtime observation and are outside the scope of pre-merge structural analysis.
- **Constraint design.** Decision Analysis consumes Constraints as given; it provides feedback on constraint adequacy (Section 6.3) but does not prescribe constraint content.

9. Conclusion

Decision Analysis addresses a structural gap in AI-assisted software development: the absence of any mechanism that determines whether the output of a generation event falls within its authorized scope.

The framework introduces Decision Effect as a first-class analytical primitive, extracts it from its implicit distribution across testing, monitoring, and review, and uses it as the entry point for a scope authorization analysis based on three Anchor Sets: Visible Scope (what the AI was authorized to see), Artifact Scope (what the AI was expected to modify), and Outcome Scope (what the AI actually modified).

Five mutually exclusive analytical states — DA-N, DA-O, DA-I, DA-V, DA-U — are defined as set-relational conditions over these three Scopes. A diagnostic protocol enables automated pre-merge scope audit, review triage, cumulative scope tracking, and constraint quality feedback.

The framework inherits its structural vocabulary from Anchor Architecture, Viewpoint-Structured Specification, and Boundary, and provides the analytical foundation upon which Decision Risk builds its governance interpretation.

Decision Analysis does not evaluate code quality. It does not assess risk. It does not prescribe governance action. It answers one question: did this generation event stay within its authorized scope? In AI-assisted environments where production rate exceeds review capacity, where generation is stateless but systems are stateful, and where correct outcomes can mask unauthorized scope expansion, this question has no existing answer. Decision Analysis provides one.

10. Related Work

Decision Analysis is positioned at the intersection of AI-assisted code generation governance, structural traceability, and effect-oriented audit. While the framework introduces novel constructs (Decision Effect, three Anchor-based Scopes, and the five mutually exclusive DA states), several strands of external literature address related challenges such as scope creep in autonomous agents, loss of decision provenance, PDCA-style governance loops, and traceability-by-product in AI-augmented workflows. No prior work provides the exact combination of effect-first structural analysis, set-relational decidability over $(\mathcal{A}, R)$, or pre-merge scope audit grounded in Anchor Architecture + VSS + Boundary.

10.1 Scope Creep and Aggressive Implementation in AI Code Agents

FeatBench (Chen et al., arXiv:2509.22237, 2025) is the closest empirical counterpart to the phenomena formalized in Decision Analysis. The benchmark evaluates repository-level feature implementation using purely natural-language requirements (no code hints) across diverse domains. Its evaluation of state-of-the-art agent frameworks reveals that “aggressive implementation” dominates failures, accounting for 73.6% of regressive cases. Agents proactively refactor or extend code beyond explicit user intent, causing widespread Pass-to-Pass (P2P) regressions. This is the observable manifestation of **Inference Creep** and the **DA-O** state ($S_o \not\subseteq S_a$). FeatBench quantifies intent misalignment via resolved rate (maximum 29.94%) but remains an offline evaluation benchmark. Decision Analysis operationalizes the same problem as an automated, pre-merge, set-membership audit that requires no test execution and operates directly on structural anchors.

10.2 Loss of Decision Traceability and Ghost Reasoning

Literature on “ghost” decisions in AI agents highlights the provenance collapse that Decision Analysis detects as **DA-V** (Decision Vacancy / Ghost Intent). McNamara (2026) describes how agents produce rich reasoning traces (trade-offs, scope markings, architectural choices) during a session but discard them upon commit, leaving only opaque code changes. The blog post “The Ghost in the Commit” explicitly calls for persistent context graphs to retain such traces for cross-session governance. Decision Analysis provides the complementary structural detection criterion: when $trace^{-1}(S_o \setminus S_a) \cap \mathcal{A}_{known} = \emptyset$, the effect exists but the originating decision does not. The framework makes Ghost Intent decidable without requiring heuristic graph construction.

10.3 PDCA Governance Loops for AI Code Generation

The **PDCA framework for AI code generation** (Judy, InfoQ, October 2025) structures human-AI collaboration into explicit Plan (scope definition), Do (generation), Check (pre-merge completeness against plan, regression detection), and Act (micro-retrospective). It shares the PDCA governance loop with Decision Analysis and emphasizes observable success criteria per session. However, Judy’s Check phase relies on narrative checklists and human judgment. Decision Analysis supplies deterministic set-relational conditions over three Scopes, producing machine-readable DA states for review triage and constraint feedback. Thus, DA can serve as a structural automation layer within Judy’s PDCA cycle.

10.4 Traceability-by-Product and Compliance-Ready Frameworks

Agile V (Koch et al., arXiv:2602.20684, 2026) merges Agile iteration with V-Model verification into a continuous “Infinity Loop” executed at task level by specialized AI agents (Requirement Architect, Build Agent, Test Designer, Compliance Auditor). Every cycle generates a traceability matrix, decision-

rationale logs, and regulatory artifacts as a by-product, enforced by human approval gates. This achieves “audit-ready delivery” on bounded projects. Agile V shares Decision Analysis’s emphasis on structural traceability and pre-merge verification; its traceability matrix is functionally analogous to $trace^{-1}$ operations. The primary distinction is that Agile V is a full workflow orchestration framework focused on regulatory compliance (ISO 9001, GxP), while Decision Analysis is a pure analytical layer that classifies effects into five governance-neutral states.

10.5 Decision-Driven Paradigms and Broader Governance

Decision-Driven Software Engineering (DDSE) (Manna, Zenodo:16462014, 2025) proposes a new SDLC paradigm that treats technical decisions as first-class artifacts to preserve human agency under AI assistance. DDSE recognizes the opacity of AI-generated decisions and advocates decision records as the system of record. However, it does not formalize scopes, effects, or set-relational states. Similarly, broader AI governance literature emphasizes boundaries and audit trails but operates primarily at the policy/risk-interpretation layer—the exact downstream interface that Decision Analysis explicitly delegates to the companion Decision Risk paper.

10.6 Summary of Coverage

The external literature converges on the symptoms Decision Analysis formalizes: aggressive scope exceedance (FeatBench), lost intent provenance (McNamara), need for structured governance loops (Judy), and traceability artifacts (Agile V). Decision Analysis synthesizes these concerns into a minimal, decidable, infrastructure-grounded framework that requires only the three upstream structures and operates without semantic code understanding. It thereby supplies the missing structural “Check” for scope authorization that no prior practice has provided in this exact form.

Response:

Companion Papers in the Present Research Series

Tsai, S. (2026). Viewpoint-Structured Specification: A Framework for Structuring Intent into Traceable Specifications as the Basis for AI-Assisted Code Generation. engrXiv / Zenodo.

Tsai, S. (2026b). From Inference Creep to Risk Acceleration Pipelines: Decision Vacancy and Governance Risks in AI-Assisted DevOps. SSRN. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=6146686

Tsai, S. (2026e). Anchor Architecture: A Minimal Structural Foundation for Software Traceability in AI-Assisted Software Development. engrXiv / Zenodo. https://doi.org/10.5281/zenodo.18856781

Tsai, S. (2026f). Boundary as an Execution-Time Primitive for AI-Assisted Software Development Governance. engrXiv / Zenodo. https://doi.org/10.5281/zenodo.18883242

Tsai, S. (2026g). Ghost Intent: An Effect of Traceability Collapse in GenAI-Assisted SDLCs. Zenodo. https://doi.org/10.5281/zenodo.18872540

Tsai, S. (2026h). A Taxonomy of Degradation Phenomena in AI-Generated Code. *Companion work in preparation.*

Tsai, S. (2026). Decision Risk: Governance Interpretation of Decision Analysis Undecidability. SSRN. *Companion work.*

External Literature

Agile V: A Compliance-Ready Framework for AI-Augmented Engineering — From Concept to Audit-Ready Delivery Koch, C. (2026). arXiv:2602.20684 [cs.SE]. https://arxiv.org/abs/2602.20684

A Plan-Do-Check-Act Framework for AI Code Generation Judy, K. (2025, October 20). InfoQ. https://www.infoq.com/articles/PDCA-AI-code-generation/

Decision-Driven Software Engineering (DDSE) for AI-Assisted Development — A New SDLC Paradigm Manna, M. R. (2025). Zenodo. https://zenodo.org/records/16462014 (Also available at: https://ai.gopubby.com/decision-driven-software-engineering-ddse-for-ai-assisted-development-the-foundational-419b93518c31)

FeatBench: Evaluating Coding Agents on Feature Implementation for Vibe Coding Chen, H., Li, C., & Li, J. (2025). arXiv:2509.22237 [cs.SE]. https://arxiv.org/abs/2509.22237 (Updated version: February 2026)

The Ghost in the Commit: Governing AI Agents That Build Our Infrastructure McNamara, C. (2026, January 6). https://colinmcnamara.com/blog/ghost-in-the-commit-governing-ai-agents-building-infrastructure