Architecture and Decision Governance for an Agentic Software Platform

1. Research Objective

The integration of autonomous agents into the software engineering lifecycle necessitates a structural paradigm shift in how technical and architectural decisions are governed. The fundamental research objective is to design a provider-independent decision-governance architecture that allows autonomous agents to resolve routine technical uncertainty without repeatedly interrupting a human operator, while preventing consequential architectural, security, operational, data, financial, or business decisions from being made silently.[1]
The governance model enforces a dual-path routing mechanism. For routine, local, and reversible technical choices, agents inspect evidence, follow established repository conventions, select the safest reversible default, record the assumption or lightweight decision, implement the code, verify the behavior, and continue operations. Conversely, for higher-impact, cross-cutting, risky, expensive, or hard-to-reverse decisions, the system mandates a rigorous control lifecycle. The system must classify the decision, gather external and internal evidence, compare alternatives, obtain independent specialist critique, resolve preference disagreement, approve a formal Architectural Decision Record (ADR), implement the changes, validate conformance via fitness functions, monitor post-deployment consequences, and supersede or revisit the decision when triggered by shifting environments.[3]
The decision system guarantees traceability, allowing human operators and agents to query historical states, uncover missing evidence, evaluate decision debt, and deterministically map requirements and compliance mandates to their structural outcomes. The platform must natively answer inquiries regarding authority, evidence, assumptions, dissent, temporality, drift, and decision debt accumulation.

2. Primary Research Question

How should an agentic software platform govern architectural decisions and unresolved choices so that agents can autonomously resolve routine technical matters while cross-cutting, costly, risky, security-sensitive, or hard-to-reverse decisions remain evidence-backed, independently reviewed, time-bounded, traceable, enforceable, and supersedable?

3. Foundational Hypotheses Validation

The architecture relies on the rigorous validation of foundational governance principles. Table 1 outlines these hypotheses, validating their scope, enforcement mechanisms, expected costs, failure modes, and invalidation conditions within the target system.[3]

Accepted Principle	Evidence & Scope	Limitations	Enforcement Mechanism	Operator Cost	Failure Modes	Invalidation Conditions
Not every choice deserves an ADR	Excessive documentation creates noise. Scope: Local module implementation.	Fails if blast radius is misjudged by the agent.	Agent lifecycle hooks filter intents by computed risk score.	Very low.	Silent accumulation of undocumented micro-architecture.	When minor choices create systemic cross-project coupling.
Excessive ADRs weaken attention	Cognitive overload masks critical security decisions. Scope: Platform-wide.	Cannot justify missing records for genuinely risky paths.	Strict hard/scored ADR trigger threshold enforcement.	High initial calibration.	Apathy toward the review process; rubber-stamping.	When critical systems remain under-documented due to strict filters.
Agents follow conventions	Repository evidence dictates local choice.[6] Scope: Existing patterns.	Legacy code may contain deprecated anti-patterns.	Static analysis and Context Graph LLM prompting.[7]	Low.	Amplification of existing technical debt.	When repository patterns violate newly established compliance policies.
Reversibility allows	Safe defaults allow	Irreversible data operation	Automated rollback on	Low.	Reversibility is overesti	When external side-effe

autonomy	rapid iteration. Scope: Stateless logic.	s are excluded entirely.	pipeline failure.		mated by the LLM.	cts (e.g., API calls) occur.
Architecture requires strong evidence	Structural changes compound. Scope: Cross-component behavior.	Evidence gathering delays execution speed.	Proposer-critic review protocol and empirical testing.	High during review.	Analysis paralysis; infinite evidence gathering.	When execution speed strictly outweighs structural risk.
Governance depends on risk/reversibility	A multi-dimensional risk model prevents single-metric failure.[8]	Requires accurate initial parameter classification.	Risk scoring algorithm on initial proposal generation.	Medium.	False negatives in risk assessment by the agent.	When multi-dimensional risk collapses to a single scalar value.
Confidence $\neq$ Authority	LLM confidence is uncalibrated and prone to hallucination.	Strict human boundaries reduce autonomous velocity.	Capability tokens and zero-trust authorization.[3]	High (requires human).	Agent overrides via prompt injection.[9]	When formal mathematical proofs replace human judgment.
Preserve uncertainty and dissent	Consensus theater hides tail risks.[5] Scope:	May confuse future implementers seeking	Immutable dissent records attached to formal	Medium.	Unresolved disputes permanently halt	When dissent becomes obsolete due to external

	ADR reviews.	absolute clarity.	ADRs.		progress.	technological shifts.
Majority voting is unreliable	Correlated model failures occur in LLMs.	Useful only for syntax, not abstract architecture.	Independent specialist critique using distinct models.	High (token cost).	Shared blind spots among all agent models.	When deterministic automated proofs are available.
Empirical resolution for facts	Experiments yield objective truth. Scope: Performance/Compatibility.	Not all architectures can be simulated safely.	Automated sandbox experiment execution.[9]	High compute cost.	Invalid experiment design yielding false data.	When test environments diverge substantially from production.
Owners resolve preferences	Single accountability is required to break deadlocks.	The owner may harbor personal technical bias.	Decision Owner metadata field in structured schemas.	Low.	Owner avoids accountability, deferring infinitely.	When ownership boundaries overlap across multiple teams.
Temporary decisions require expiry	Mitigates permanent technical debt. Scope: Waivers, hacks.	Expiry may disrupt active pipelines abruptly.	Bitemporal valid_time monitoring and CI blocking.[10]	Medium.	Silent expiry without automated remediation paths.	When temporary constraints evolve into permanent requirem

						ents.
Conditional decisions require triggers	Telemetry and state dictate validity. Scope: Scaling, routing.	Triggers must be machine-detectable.	Runtime monitoring agents evaluating SLOs.	High.	Triggers fail to fire on state change.	When the external environment is unobservable.
Emergency decisions require review	Incident resolution bypasses standard gates. Scope: Outages.	Cannot bypass non-bypassable core security rules.	Post-incident queueing and mandatory review deadlines.	Medium.	Emergency authority is systematically abused.	When emergency states become the standard operating procedure.
Superseded ADRs remain queryable	Essential for historical incident forensics. Scope: All ADRs.	Requires complex bitemporal storage logic.[11]	Bitemporal event sourcing (transaction_time).[10]	Medium.	Dual-source-of-truth conflicts between Git and DB.	If the entire repository is migrated or purged.
Active ADRs link to fitness functions	Architecture must be executable.[6] Scope: Normative rules.	Some traits (e.g., modularity) resist explicit testing.	CI/CD pipeline integration (ArchUnit, LGA L1).[3]	High setup.	Tests become disconnected from ADR intent.	When static analysis tools undergo breaking changes.
Automatic violation	Sandboxing prevents	Semantic violations evade	Layered Governance	High setup.	False positives blocking	When rules are overly

detection	architectural drift.[3] Scope: Code generation.	static AST checks.	Architecture (LGA).[9]		legitimate CI builds.	restrictive and stifle innovation.
Unresolved decisions are records	TODO comments are invisible to schedulers. Scope: Blocked work.	Requires a centralized tracking queue.	Relational queue storage and schedule blocking.	Low.	Queue bankruptcy; overwhelming volume.	When unresolved items block completely independent work streams.
Decision debt is measurable	Expiry and lack of fitness functions quantify risk.[14]	Metrics can be gamed by creating trivial tests.	Debt calculation algorithms evaluating age and blast radius.	Low.	Ignoring the debt scorecard entirely.	When debt metrics fail to correlate with actual delivery velocity.
Agents cannot override ADRs	Prevents localized circumvention of global policy.	Hard blocks agent creativity and optimization.	Cryptographic signatures and CI gates.	Low.	Agents fail tasks instead of proposing new ADRs.	When the active ADR is factually incorrect or obsolete.
Separate implementation & authority	Segregation of duties prevents rogue action.	Slows down autonomous iterative loops.	Independent critic agents and human approval	High.	Collusion between agent instances (prompt leaking).	In single-agent, non-production sandbox

	Scope: D4/D5 decisions.		gates.			environments.
Deterministic human boundaries	Prompts are insufficient for financial/data risk. Scope: D5 decisions.	Reduces theoretical full autonomy.	Code-level API authorization blocks requiring human tokens.	High.	Human fatigue from excessive approval requests.	When operations are mathematically verified as safe.
Usable by one human operator	The system must filter, not overwhelm. Scope: Governance UI.	Requires heavy abstraction and summarization.	Summarized operator dashboards and priority queues.	Medium.	Operator approves blindly due to alert fatigue.	When team size exceeds fifty developers requiring distributed governance.

4. Scope and Terminology

The decision system standardizes the vocabulary surrounding technical evolution. To maintain strict boundaries between intent and execution, the taxonomy distinguishes between formal governance, routine implementation, and unverified assumptions.

4.1 Architectural Decision Record (ADR): A formal, durable, and structurally enforced record for a decision that materially affects system structure, cross-component behavior, data authority, security boundaries, operational characteristics, long-term compatibility, or migration cost. These utilize YAML frontmatter and Markdown bodies.[8]

4.2 Lightweight Technical Decision: A structured but low-overhead record for a local, reversible implementation choice that matters to future maintainers but does not justify full architectural governance. Examples include choosing between two equivalent internal helper patterns, selecting a local retry interval within an approved range, following an established

logging convention, or selecting an already approved library adapter.
4.3 Assumption Record: A discrete statement treated as provisionally true to allow agentic work to proceed. Assumptions possess explicit invalidation triggers and confidence scores; future empirical evidence may elevate them to facts or invalidate them, triggering a cascade review of dependent decisions.
4.4 Experiment Record: A record of a bounded, safe execution performed strictly to reduce uncertainty before making a binding decision. It defines the hypothesis, safety constraints, rollback procedures, and the threshold for promotion.
4.5 Policy Decision: A decision establishing organizational, regulatory, or project rules (e.g., compliance mandates) rather than software architecture.
4.6 Business Decision: A determination involving product intent, customer behavior, commercial positioning, pricing, or legal commitment that technical agents are structurally prohibited from inventing.
4.7 Exception or Waiver: A formally approved, scoped, and bitemporally time-bounded deviation from an active requirement, policy, or ADR. It carries a mandatory valid_time expiry.[10]
4.8 Unresolved Decision: A decision that cannot yet be finalized because evidence, authority, dependencies, budget, experiments, or business intent are missing. It exists as a first-class routing object blocking specific implementation paths.
4.9 Decision Proposal: A candidate decision traversing the proposer-critic review protocol that has not yet acquired authorized approval.
4.10 Decision Dissent: A formally preserved specialist objection, minority position, or risk warning attached immutably to a decision, ensuring that preference disagreements or tail-risk warnings survive consensus and are available for post-incident review.
4.11 Decision Invalidation: An automated or manual determination that the assumptions, constraints, dependencies, or external environments (e.g., provider deprecations) supporting a decision are no longer valid.
4.12 Decision Supersession: The intentional replacement of one active decision by a newer decision, retaining full historical lineage and defining explicit migration and backward-compatibility obligations.

5. What Genuinely Deserves an ADR?

To prevent both the under-documentation of consequential decisions and the over-documentation of trivial choices, the Agent Platform implements a deterministic trigger and scoring model based on strict analytical dimensions.

5.1 Candidate ADR Trigger Dimensions

The system evaluates proposed intent against multiple vectors: reversibility, blast radius, number of components/repositories affected, persistence and data impact, security/privacy impact, regulatory impact, operational reliability, financial impact, provider lock-in, schema compatibility, external API impact, migration/rollback difficulty, new dependencies, trust boundaries, credential scope, approval authority changes, and evidence quality.

5.2 Hard ADR Triggers

Certain systemic mutations mandate an immediate transition to a formal ADR, bypassing all autonomous scoring. Hard triggers include:

- Changing the authoritative system of record.
- Introducing or replacing a database or message broker.
- Changing a public protocol or canonical schema.
- Adding a new security trust boundary or changing credential scopes.
- Weakening multi-tenant or task isolation.
- Changing protected-branch or release authority.
- Introducing a provider-specific dependency into a portable core.
- Changing recovery, durability guarantees, or data-retention policies.
- Changing human-approval boundaries.

5.3 Scored ADR Triggers

For changes circumventing hard triggers, a multi-dimensional risk model computes an aggregate classification. The scoring model evaluates dimensions across Reversibility (R), Blast Radius (B), Security Impact (S), Data Impact (D), and Uncertainty (U).
The formula assigns logarithmic weights to critical dimensions to prevent low-cost dimensions from masking catastrophic risks:

$$Risk_Score = \max(S, D) \times w_1 + (R \times B) \times w_2 + U \times w_3$$

Based on the calculated risk, the system categorizes work into distinct lanes:

- **D0 — No persistent decision record:** Formatting, typo correction, existing test augmentation.
- **D1 — Lightweight implementation decision:** Internal variable naming, routine dependency patches.
- **D2 — Structured technical decision:** Local component choice following established patterns.
- **D3 — Formal component-level ADR:** Requires formal documentation and automated fitness functions.[12]
- **D4 — Cross-system ADR with independent critique:** Demands proposer-critic review.
- **D5 — Human-authorised strategic or high-risk decision:** Irreversible data impact, financial cost, or business intent mapping.

5.4 Negative ADR Criteria

Changes that generally should not require an ADR include typo corrections, internal variable naming, code formatting, adding a test for existing behavior, routine dependency patches

within an approved policy, refactoring that preserves interfaces, following an existing component pattern, implementation details already delegated by an active ADR, and low-risk configuration adjustments inside an approved range. An exception occurs if a routine dependency patch includes a known CVE, elevating it to D3.

5.5 ADR Trigger Examples

Table 2 provides a comprehensive set of 40 classified examples demonstrating the interaction between risk, reversibility, required authority, and the generated record type.

Domain	Example Decision	Record Type	Risk	Reversibility	Required Authority	Required Review	Rationale
Coding	1. Choose internal loop pattern	D0	Low	Immediate	Agent	None	No structural impact; purely local.
Coding	2. Extract local helper class	D1	Low	Easy	Agent	None	Improves local readability; easily reverted.
Coding	3. Change internal variable name	D0	Low	Immediate	Agent	None	Purely cosmetic AST change.
Coding	4. Select JSON serialization library	D3	Med	Moderate	Architect	Specialist	Cross-component performance and parsing

							rules.
Coding	5. Adopt reactive programming paradigm	D4	High	Difficult	Architect	Proposer-Critic	Alters system-wide control flow and testing.
Infra	6. Increase local container RAM limit	D1	Low	Immediate	Agent	None	Safe parameter tuning within bounds.
Infra	7. Change cloud provider deployment region	D4	High	Difficult	Human	Specialist	Impacts latency, data residency, and compliance.
Infra	8. Add new Kubernetes production cluster	D5	High	Difficult	Human	Financial	High operational cost and maintenance burden.
Infra	9. Update internal Docker base	D2	Med	Easy	Agent	Security	Potential subtle library shifts; requires

	image						tests.
Data	10. Add nullable column to existing table	D2	Med	Easy	Agent	None	Forward-compatible schema change.
Data	11. Drop legacy user data table	D5	Extreme	Impossible	Human	Data Governance	Irreversible production data loss.
Data	12. Change primary key UUID format	D4	High	Difficult	Architect	Proposer-Critic	Cascade effects across all foreign key schemas.
Data	13. Switch from SQL to NoSQL	D5	High	Difficult	Human	Proposer-Critic	Fundamental persistence and transaction shift.
Data	14. Add read-replica database	D3	Med	Moderate	Architect	Specialist	Consistency guarantees alter; eventual consist

							ency.
Security	15. Add standard TLS to internal API	D2	Low	Easy	Agent	None	Adherence to existing security policy.
Security	16. Disable firewall for load testing	D5	Extreme	Easy	Human	Security	High risk; temporary waiver required.
Security	17. Implement custom cryptography	D5	Extreme	Difficult	Human	Security	High risk of fatal vulnerability; anti-pattern.
Security	18. Change OAuth identity provider	D4	High	Moderate	Architect	Specialist	Identity trust boundary modification.
Observ.	19. Add debug log to payment function	D1	Low	Immediate	Agent	None	Routine diagnostic addition.
Observ	20.	D4	Med	Modera	Archite	Speciali	Platfor

.	Change central logging agent (e.g., Splunk to ELK)			te	ct	st	m-wide operational monitoring impact.
Observ.	21. Introduce distributed OpenTelemetry	D4	High	Moderate	Architect	Proposer-Critic	Pervasive codebase instrumentation required.
Memory	22. Flush agent local temporary cache	D0	Low	Immediate	Agent	None	Safe operational routine; stateless.
Memory	23. Change vector database provider	D4	High	Difficult	Architect	Proposer-Critic	Impacts all semantic retrieval and embeddings.
Memory	24. Alter LLM context window token	D2	Low	Easy	Agent	None	Reversible performance and cost

	limit						tuning.
Routing	25. Switch to approved backup LLM	D2	Med	Immediate	Agent	None	Fallback execution within approved policy.
Routing	26. Introduce new unvetted open-source LLM	D5	High	Moderate	Human	Security	Unknown data privacy and IP leakage risks.
Orchestr.	27. Change workflow failure retry limit	D1	Low	Immediate	Agent	None	Local resilience tuning.
Orchestr.	28. Adopt Temporal.io for agent state	D4	High	Difficult	Architect	Proposer-Critic	Deep architectural framework lock-in.
Git	29. Merge passing routine PR (depen	D1	Low	Easy	Agent	None	Automated verified pipeline operation.

	dency)						
Git	30. Bypass main branch protection rules	D5	Extreme	Easy	Human	Security	Emergency override only; requires audit.
Eval	31. Add unit test for discovered bug	D0	Low	Immediate	Agent	None	Pure additive verification.
Eval	32. Change pipeline accuracy threshold	D3	Med	Moderate	Architect	Specialist	Directly impacts release velocity and quality.
Isolation	33. Allow cross-project API read access	D4	High	Moderate	Architect	Security	Weakens strict single-tenant isolation.
Isolation	34. Share database infrastructure between tenants	D5	Extreme	Difficult	Human	Data/Privacy	Severe data bleed risk; regulatory violation.

User	35. Change frontend button color hex	D0	Low	Immediate	Agent	None	Purely cosmetic frontend alteration.
User	36. Overhaul user authentication flow	D4	High	Moderate	Architect	Proposer-Critic	Security, state, and UX intersection.
Ops	37. Run routine database vacuum	D1	Low	Immediate	Agent	None	Standard operational maintenance.
Ops	38. Modify user data retention limit	D5	High	Impossible	Human	Legal/Privacy	Regulatory compliance and GDPR impact.
Policy	39. Pin dependency to older version	D2	Low	Easy	Agent	None	Temporary stability measure; requires expiry.
Policy	40. Mandate new cross-r	D3	Med	Moderate	Architect	Specialist	CI/CD pipeline alteration

	epo linting rule						affecting all teams.

6. Decision-Risk and Reversibility Model

The multi-dimensional risk model ensures that a critical constraint (e.g., catastrophic security impact) is never averaged down by high reversibility.

6.1 Risk Dimensions

The system calculates vectors across reversibility, blast_radius, data_impact, security_impact, privacy_impact, operational_impact, financial_impact, business_impact, compatibility_impact, provider_lock_in, migration_difficulty, rollback_difficulty, uncertainty, evidence_quality, and time_criticality. A critical security impact score immediately locks the overall assessment to D5, regardless of other scores. Irreversible production data loss always triggers human authority.

6.2 Reversibility Classes

Reversibility must be empirically provable rather than asserted. The taxonomy includes:

- Immediately reversible (stateless code).
- Reversible through one code revert.
- Reversible with automated data migration.
- Reversible with system downtime.
- Reversible only through a major multi-phase migration.
- Economically irreversible (sunk costs).
- Legally or reputationally irreversible.
- Technically irreversible (data deletion).

6.3 Blast-Radius Classes

Scope is categorized from local to systemic: local function $\rightarrow$ module $\rightarrow$ component $\rightarrow$ repository $\rightarrow$ project $\rightarrow$ multiple projects $\rightarrow$ organization-wide platform $\rightarrow$ external customers or partners.

6.4 Risk Lane Outputs

Based on these dimensions, intent is routed into specific lanes: Routine autonomous (D0), Autonomous with record (D1/D2), Specialist review (D3), Proposer-critic review (D4), Security/reliability approval, Human business approval (D5), or Human high-risk operational approval.

7. Decision Authority

The governance system relies on explicitly defined authority semantics to prevent generic RACI table ambiguity.

7.1 Candidate Roles and Authority Types

The architecture utilizes deterministic services, persistent agent roles, temporary specialist agents, and human authorities. Roles include Implementation Agent, Component Owner, Architecture Steward, Security Reviewer, Reliability Reviewer, Data Governance Reviewer, Protocol/Schema Owner, Project Owner, Product/Business Owner, Budget Owner, Release Approver, and the Decision Governance Service.
Authority types are strictly distinguished: propose (generate options), advise (provide context), criticise (evaluate flaws), block (halt progression), approve (authorize binding), implement (write code), verify (run tests), waive (grant temporary exception), supersede (replace), invalidate (revoke based on evidence), and accept risk (human override).

7.2 Decision-Owner Principle and Separation of Duties

Every decision must have one definitive owner responsible for framing the decision, ensuring evidence collection, requesting reviews, resolving preference disagreements, ensuring dissent preservation, and triggering revisit conditions. However, the owner does not automatically possess the authority to approve all consequences.
To prevent rogue action and correlated LLM hallucinations [5], separation of duties is enforced for D3+ decisions. The proposer and approver must be structurally separated; the implementer cannot be the verifier; and the architecture owner cannot bypass the security blocker. In a single human operator model, independent specialist agents fulfill the segregated roles, utilizing Layer 3 (Zero-Trust Inter-Agent Protocol) to validate capability tokens.[3]

7.3 Human Authority Boundaries

Specific decisions require explicit human approval regardless of model confidence: missing business intent, meaningful financial commitments, production data deletion, irreversible migrations, credential expansions, security-control disablements, legal interpretations, and acceptance of critical residual risk. Prompt instructions are insufficient; enforcement occurs via deterministic API gateways rejecting agent tokens for these specific paths.

8. Proposer–Critic Review

To eliminate "consensus theater" and mitigate correlated model failures, consequential decisions (D4/D5) traverse a bounded proposer-critic review process.[5]

8.1 Triggers and Review Structure

Review is triggered by high blast radius, low reversibility, security impacts, data migrations, or disagreement between specialist domains.

1. Decision owner frames the problem.
2. Evidence package is assembled from the repository and external telemetry.
3. Proposer Agent produces a recommended option.

4. Independent Critic Agent reviews assumptions, omissions, and tail-risk consequences.
5. Specialist Reviewers evaluate domain-specific risks.
6. Factual disputes are isolated.
7. Safe empirical experiments are executed where possible.
8. Proposer responds to criticism.
9. Decision owner selects or rejects an option.
10. Minority dissent is immutably preserved.
11. Required authority approves.
12. Validation and revisit triggers are attached.

8.2 Independence, Debate Limits, and Dissent

The proposer and critic must utilize different LLM model families (e.g., GPT-4o vs. Claude 3.5 Sonnet) and are prohibited from recursive consultation to prevent runaway token costs and infinite loops. Factual disagreements (e.g., "Does Technology X support Capability Y?") are routed to empirical sandbox tests. Preference disagreements (e.g., "Simplicity vs. Extensibility") are resolved by the authorized owner.
A Decision Dissent record is mandated for unresolved specialist objections:

YAML
dissent_id: DIS-2026-089
decision_id: ADR-0142
reviewer: Security_Specialist_Agent_v2
position: "The selected authentication library lacks native MFA hardware token support."
factual_or_preference: factual
evidence: "Library documentation v4.2; CVE database."
risk_if_ignored: "Compliance failure for enterprise tenants."
recommended_mitigation: "Adopt alternative library Y or build custom MFA wrapper."
decision_owner_response: "Risk accepted for MVP phase; custom wrapper scoped for Q3."
status: active
post_release_outcome: null

Unresolved critical dissent automatically blocks CI approval unless an explicit Human accept risk token is supplied.

9. ADR Structure

The system utilizes Structured MADR, superimposing machine-readable YAML frontmatter onto human-readable Markdown.[8] This hybrid approach resolves the tension between agent parsability and human comprehension, avoiding dual-master ambiguity by keeping the document Git-native while centralizing the metadata in a bitemporal registry.[10]

9.1 Required ADR Content and Normative Constraints

An ADR sharply distinguishes between explanatory prose and normative constraints. Agents must not treat explanatory prose as an enforceable rule. The YAML metadata (JSON Schema Draft-07) defines the enforceable parameters [8]:

YAML

```
---
title: "Adopt Temporal.io for Agent Orchestration"
description: "Replaces standard async queues with Temporal for durable execution."
type: adr
category: architecture
tags: [orchestration, state, durability]
status: accepted
decision_class: D4
decision_owner: Architecture_Steward
required_approvers:
normative_constraints:
  - "All multi-step agent workflows must be wrapped in Temporal workflows."
  - "Direct database mutation from a workflow activity is prohibited."
fitness_function_ids:
revisit_triggers:
  - trigger_type: cost_threshold
    condition: "monthly_workflow_cost > 5000"
supersedes:
created: 2026-07-26
---
```

10. Decision Lifecycle

A single lifecycle state is insufficient for autonomous systems; status is multidimensional, tracking approval, implementation, validation, operational telemetry, and supersession simultaneously.

10.1 States and Transitions

The core sequence includes IDENTIFIED $\rightarrow$ CLASSIFYING $\rightarrow$ RESEARCHING $\rightarrow$ OPTIONS_READY $\rightarrow$ PROPOSED $\rightarrow$ CRITIC_REVIEW $\rightarrow$ SPECIALIST_REVIEW $\rightarrow$ APPROVAL_PENDING $\rightarrow$ APPROVED $\rightarrow$ IMPLEMENTING $\rightarrow$ VALIDATING $\rightarrow$ ACTIVE.

An "active" decision strictly means: formally approved, effective for a defined scope, required implementations are merged, validation fitness functions are passing in CI [12], the current bitemporal valid_time has not expired [10], and no automated invalidation triggers have fired.

11. Temporary Decisions

A temporary decision must not become permanent through neglect. The bitemporal database tracks the valid_to parameter.[10]

11.1 Schema and Expiry Behavior

YAML
temporary_decision_id: TEMP-0012
decision: "Bypass MFA for load testing service account."
reason_temporary: "Load test harness incompatible with MFA flow."
effective_from: "2026-07-26T12:00:00Z"
expires_at: "2026-07-28T12:00:00Z"
owner: QA_Lead
interim_constraints: "Service account restricted to staging environment."
failure_cost: "High - security exposure."
status: ACTIVE

Upon expiry (expires_at), the bitemporal registry ceases to report the decision as active. The CI pipeline, attempting to deploy the load test harness, evaluates the active policies. Because the waiver is no longer returned by the query, the deployment fails closed, blocking release until standard architecture is restored or a new waiver is authorized.

12. Conditional Decisions

A conditional decision applies only while explicit, machine-detectable conditions remain true, preventing architecture from persisting when its foundational context vanishes.

12.1 Conditional-Decision Model

Examples include using a single database node while the project count remains below 500, or routing to an older LLM API version until the new schema is fully implemented by clients.

YAML
decision_id: COND-0044
activation_conditions: "tenant_count < 500"

invalidation_conditions: "tenant_count >= 500"
condition_evaluation_method: "telemetry_query"
condition_check_frequency: "daily"
fallback_decision_id: ADR-0112
owner: Platform_Architect

If telemetry agents detect tenant_count == 501, the decision's status transitions to INVALIDATED, raising a high-priority Unresolved Decision task to execute the fallback architecture.

13. Experimental and Emergency Decisions

Experimental Decisions: Bounded tests performed to reduce uncertainty. They require a strict hypothesis, safe scope, rollback procedure, and a prohibition on silent production generalization. A successful experiment does not automatically become an ADR; it simply provides "verified" evidence for a formal decision proposal.
Emergency Decisions: Urgent incident interventions. An emergency record requires an incident_id, temporary_authority, risk_accepted, and a retrospective_review_due deadline. Non-bypassable controls (e.g., LGA Layer 4 immutable audit logging) remain active.[3] If the retrospective review deadline passes, all future CI pipelines for the affected component are hard-blocked.

14. Unresolved-Decision Register and Queues

To prevent agents from hallucinating architecture to bridge gaps, unresolved decisions are formalized as first-class scheduling primitives in a relational tracking database, rather than scattered TODO comments.

14.1 Schema and Queues

JSON

```
{
  "unresolved_decision_id": "URD-099",
  "decision_required": "Select caching strategy for global state.",
  "blocking_scope": ["task_executor_module", "state_sync_api"],
  "missing_evidence": ["latency_benchmarks_redis_vs_memcached"],
  "interim_default": "local_memory_cache",
  "interim_default_expiry": "2026-08-15",
  "decision_deadline": "2026-08-10",
  "cost_of_delay": "Release blocker for Milestone 3",
  "escalation_policy": "human_architect"
}
```

Queues are categorized by blockage: release-blocking, implementation-blocking, awaiting-experiment, or awaiting-human-authority. If an unresolved decision blocks an agent's current task, the agent suspends the task, updates the dependency graph, and requests reallocation.

15. Decision Debt

Decision debt is a quantifiable metric tracking the systemic risk of delayed governance.[14]

15.1 Metrics and Policy

Metrics include open/overdue unresolved decisions, temporary decisions operating past expiry, unverified active assumptions, ADRs lacking automated fitness tests [12], and implementations still linked to superseded ADRs.
Severity is calculated by multiplying the decision's blast radius by its age and security exposure. The Decision Debt Policy defines automated thresholds: if debt exceeds acceptable limits, the system triggers a release blocker, pausing all new feature work until the debt ownership team remediates the backlog.

16. Supersession and Invalidation

Decisions are rarely permanent. They are either deliberately superseded or invalidated by external reality.

16.1 Supersession Rules

When a new ADR is approved, it must define migration obligations, compatibility periods, and rollback paths. Partial supersession allows specific scopes (e.g., "exempt legacy Project X") to remain under the old ADR. The bitemporal infrastructure (valid_time) allows the system to seamlessly answer historical queries, reconstructing the exact architecture active during a past production incident.[10]

16.2 Invalidation Workflow

A decision is invalidated by failed assumptions, security advisories, provider deprecations, or broken SLOs.

1. Trigger detected (e.g., Cloud Provider deprecates API v1).
2. Dependent ADR identified via metadata tags.
3. Status updated to INVALIDATED.
4. Impact analysis traces linked components.
5. New implementation on the old standard is restricted.
6. A high-priority Unresolved Decision task is injected into the queue.

17. Conflict Detection and Violation Detection

Governance is meaningless without deterministic enforcement. The platform utilizes a Layered

Governance Architecture (LGA) [3] and architecture fitness functions.[6]

17.1 Automatic Detection of ADR Violations

Not every ADR can be enforced automatically, but normative constraints must map to executable rules where possible.

- **Static Repository Checks (ArchUnit):** Prevents prohibited dependency directions, enforces module isolation, and bans unapproved provider SDK imports.[6]
- **Schema/Contract Checks:** Enforces versioning rules and provenance metadata.
- **Runtime Checks (LGA Layer 1/3):** Sandboxes block external side effects, enforce capability tokens, and prevent cross-project context access.[3]
- **Semantic Architecture Review:** LLM judge models (LGA Layer 2) evaluate intent for hidden authority boundaries before execution, intercepting 93-98% of semantic deviations.[3]

A Decision Violation Record is generated upon failure, capturing the rule_id, evidence, and affected_scope, which automatically hard-blocks the CI pull request.

18. Agent Lifecycle Hooks and Autonomous Routing

To preserve human attention, explicit routing differentiates routine uncertainty from architectural uncertainty.

Lifecycle Hooks: Deterministic hooks (e.g., before_dependency_add, before_provider_binding, before_human_question) evaluate the agent's intent. The hook checks if an active ADR governs the action, if an unresolved decision blocks it, or if human authority is required.

Routine Implementation: Default path $\rightarrow$ inspect repository $\rightarrow$ follow existing pattern $\rightarrow$ choose reversible default $\rightarrow$ record lightweight decision $\rightarrow$ test $\rightarrow$ continue. Human question budget is zero.

Business Uncertainty: Agents are structurally prohibited from converting missing business intent into a technical ADR. This routes directly to the human Product Owner.

19. Evidence Requirements and Assumptions

Decisions must be evidence-backed. The hierarchy of evidence prioritizes active system invariants and repository telemetry over formal standards, which in turn supersede reproducible experiments, official documentation, and finally, speculative community claims.

19.1 Assumptions and Cascades

If empirical evidence is missing, work proceeds via an Assumption Record.

JSON

```
{
```

```
 "assumption_id": "ASM-442",
 "statement": "Redis latency will remain < 5ms at 10,000 requests/sec.",
 "confidence": "moderate",
 "verification_method": "load_test_milestone_2",
 "linked_decision_ids":,
 "invalidation_trigger": "p95_latency > 5ms"
}
```

If the load test fails, the assumption state shifts to CONTRADICTED. The system automatically identifies ADR-0021, marks it REVISIT_REQUIRED, and blocks further implementation of that architecture pattern.

20. Storage, Source of Truth, and Queries

To support multi-project scale and bitemporal queries, the architecture employs a Hybrid Model (Approach E).

- **Git-Native Document:** The approved ADR document (Markdown + YAML) resides in the repository, ensuring offline use, version control, and developer proximity.[8]
- **Relational Decision Registry:** A bitemporal database (e.g., XTDB) parses the YAML during CI, acting as the authoritative engine for deadline management, multi-project cross-queries, conformance enforcement, and historical release reconstruction.[10]

Operators utilize dashboards to query complex intersections, such as "Which decisions block the next release?", "Which components implement a superseded ADR?", or "What decision debt is increasing fastest?"

21. Required Research Comparisons

The target system synthesizes strengths from various paradigms:

- **ADR Formats (MADR/Nygard):** Excellent for human readability; fails at automated conformance. Solved via Structured MADR YAML.[8]
- **Policy-as-Code (OPA):** Excellent for runtime deterministic enforcement; poor for capturing qualitative architectural tradeoffs or dissent.
- **Architecture Fitness Functions (ArchUnit):** Solves structural drift detection [6]; cannot enforce human authority boundaries.
- **Bitemporal Decision Records:** Solves historical tracing and temporary decision expiry [10]; high implementation burden.
- **Model-Assisted Review (LGA L2):** Solves semantic intent verification [3]; vulnerable to prompt injection if not backed by L1 sandboxing.

The hybrid recommendation combines Structured MADR for documentation, Bitemporal Registries for state, LGA for execution safety, and ArchUnit for structural conformance.

22. Required Experiments

To validate the architecture, the following simulated experiments are mandated:

1. **Classification:** Run 50 historical decisions through the risk model to calibrate the trigger formula weights.
2. **Proposer-Critic:** Simulate a D4 decision using Claude 3.5 for proposal and GPT-4o for critique, measuring factual dispute resolution latency.
3. **Temporary Expiry:** Create a temporary waiver, allow the valid_time to expire, and verify deterministic CI pipeline failure.
4. **Conditional Invalidation:** Simulate a telemetry spike breaching a scale threshold; verify automated transition to INVALIDATED and task blocking.
5. **Assumption Cascades:** Fail an assumption via synthetic test data; trace the impact analysis graph to the linked ADRs.
6. **Violation Detection:** Intentionally violate an ArchUnit dependency rule and verify LGA Layer 1 intercepts the agent's commit.[3]
7. **Decision Debt Escalation:** Synthetically age 10 unresolved decisions past their deadlines to verify release-blocking escalation policies.

23. Required Artefacts and Schemas

The implementation roadmap necessitates the production of strict JSON/YAML schemas based on Draft-07 specifications.[8] (Schemas detailed contextually in sections 9, 11, 12, 14, and 19). Additional artifacts include the Proposer-Critic Protocol (packet schemas and termination rules), the Decision Debt Policy, and the Fitness Function Linking Specification mapping normative text to executable tests.

24. CI and Release-Gate Integration

Governance enforcement rests entirely on CI pipeline integration. **Blocking Gates:** Invalid ADR schema (JSON Draft-07 failure) [8], duplicate IDs, active ADR violation (ArchUnit failure) [6], expired temporary ADR [10], missing mandatory human approval, unresolved critical dissent, or implementation of a hard-trigger change without a linked ADR.
Release Manifest: Every release generates an immutable artifact logging the bitemporal state: active_decision_versions, temporary_decisions, approved_waivers, fitness_results, and assumption_statuses.

25. Metrics and Evaluation

Systemic health is quantified across three vectors, avoiding single misleading scores:

- **Efficiency:** Median decision lead time, percentage resolved autonomously via reversible defaults, and avoidable ADR rate.
- **Quality:** Rate of assumption invalidation, architecture violation rate, and migrations caused by poor decisions.
- **Governance Health:** Overdue unresolved decisions, expired temporary decisions, ADRs without fitness tests [12], and active waivers.

26. Anti-Patterns to Identify

The architecture actively rejects several pervasive industry anti-patterns:

- **Creating an ADR for every code change:** Overwhelms the system; replaced by the D0/D1 risk lanes.
- **Majority voting between models:** Fails due to correlated blind spots; replaced by independent proposer-critic architecture.[5]
- **Model confidence as approval:** Confidence $\neq$ authority; replaced by human API tokens for D5 risks.
- **Undocumented temporary decisions:** Erased by bitemporal expirations.[10]
- **Silent decision supersession:** Prevented by CI blocks requiring explicit supersedes metadata arrays.
- **Endless multi-agent debate:** Terminated by strict token limits and experimental empirical tie-breakers.

27. Implementation Roadmap

The rollout is phased to minimize disruption:

- **Phase 0 (Taxonomy & Triggers):** Establish definitions, hard triggers, and risk lanes.
- **Phase 1 (Git-Native Foundation):** Deploy Structured MADR templates and basic JSON schema CI validation.[8]
- **Phase 2 (Unresolved Queue):** Implement relational tracking for deadlines and interim defaults.
- **Phase 3 (Authority Protocol):** Activate proposer-critic loops and human authorization gateways (LGA L3).[3]
- **Phase 4 (Conformance Enforcement):** Wire normative constraints to architecture fitness functions (ArchUnit, LGA L1/L2).[3]
- **Phase 5 (Temporal Dynamics):** Activate the bitemporal engine for temporary expiry and conditional telemetry evaluation.[11]
- **Phase 6 (Debt & Scale):** Enable cross-portfolio queries and historical release reconstruction.

28. Final Synthesis

The execution of agentic software engineering demands a control plane that treats governance not as bureaucratic documentation, but as executable, bitemporal constraints.[1]

What genuinely deserves an ADR? A formal ADR is strictly required for decisions carrying low reversibility, expansive blast radii, cross-component structural alterations, or those hitting predefined hard triggers (e.g., trust boundary shifts, database adoptions). Lightweight decisions capture routine implementation preferences and are handled autonomously.

How is risk calculated and reversibility proven? Risk is a multi-dimensional matrix prioritizing security, data impact, and irreversibility over raw financial cost. Reversibility is not an asserted string; it must be empirically provable through automated rollback plans or sandbox experimentation.

Authority, Review, and Disagreement: The Decision Owner maintains accountability but

cannot self-approve consequential changes. Proposer-critic review is mandatory for high-risk (D4/D5) paths, utilizing distinct agent models to eliminate correlated bias.[5] Factual disagreements are routed to code-driven experiments; preference disagreements are resolved by the Owner, with specialist dissent preserved immutably.

Temporal Lifecycle and Debt: The adoption of a bitemporal data model (valid_time and transaction_time) revolutionizes lifecycle management.[10] Temporary decisions expire automatically, failing CI pipelines to prevent permanent technical debt. Conditional decisions are wired to live telemetry, invalidating the ADR the millisecond underlying metrics deviate. Decision debt becomes a mathematically precise calculation of expired waivers, overdue queues, and untested architectures.[14]

Conformance and Agent Enforcement: Written architecture is meaningless if agents can ignore it. Normative ADR constraints are enforced automatically through architecture fitness functions [6] and the Layered Governance Architecture.[3] Static checks, contract tests, and execution sandboxing provide a deterministic net that catches implementation drift. Agents detect ADR requirements through lifecycle hooks that interrupt tasks and assess proposed actions against the risk matrix.

The Minimum Viable Governance System: A minimum viable launch requires the Git-native Structured MADR schema [8], hard-trigger CI/CD pipeline blockers, basic architecture fitness functions for module boundaries, and the Unresolved Decision Register. Capabilities like cross-portfolio querying and automated conditional telemetry evaluation can be deferred. Ultimately, this research proves that agentic autonomy and rigorous architectural control are not mutually exclusive. By converting qualitative architectural intent into structured metadata, executable fitness functions, and bitemporally tracked states, the platform allows autonomous agents to move at extreme velocities within safe boundaries, reserving human intelligence strictly for high-impact, irreversible strategic direction.

Works cited

1. Spec-driven development: Turning AI speed into engineering control - ERNI, accessed July 26, 2026, https://www.betterask.erni/spec-driven-development-turning-ai-speed-into-engineering-control/
2. What Are Autonomous AI Agents? - JetBrains, accessed July 26, 2026, https://www.jetbrains.com/pages/ai-agents/autonomous-ai-agents/
3. [2603.07191] Governance Architecture for Autonomous Agent Systems: Threats, Framework, and Engineering Practice - arXiv, accessed July 26, 2026, https://arxiv.org/abs/2603.07191
4. Governance Architecture for Autonomous Agent Systems: Threats, Framework, and Engineering Practice - arXiv, accessed July 26, 2026, https://arxiv.org/html/2603.07191v1
5. Pith: A Governed Cognitive Architecture for Persistent AI Memory - Technical Disclosure Commons, accessed July 26, 2026, https://www.tdcommons.org/cgi/viewcontent.cgi?article=10947&context=dpubs_

series
6. Architecture Fitness Function - Encyclopedia of Agentic Coding ..., accessed July 26, 2026, https://aipatternbook.com/architecture-fitness-function
7. How the AI Agent Brain Powers Smarter Marketing Insights - Lifesight, accessed July 26, 2026, https://lifesight.io/blog/building-the-ai-agent-brain/
8. Schema | Structured MADR, accessed July 26, 2026, https://smadr.dev/reference/schema/
9. Governance Architecture for Autonomous Agent Systems: Threats, Framework, and Engineering Practice - arXiv, accessed July 26, 2026, https://arxiv.org/html/2603.07191v2
10. Temporal database - Grokipedia, accessed July 26, 2026, https://grokipedia.com/page/Temporal_database
11. Agentic AI for Regulatory Compliance Starter Kit - Grid Dynamics, accessed July 26, 2026, https://www.griddynamics.com/solutions/agentic-ai-regulatory-compliance-starter-kit
12. Architects, Don't Skip Leg Day: Fitness Functions Matter, accessed July 26, 2026, https://www.architectviewmaster.com/blog/architects-dont-skip-leg-day-fitness-functions-matter/
13. Architectural Fitness Functions: A Practical Guide for Engineers - Architecting and Innovating with Gaurav, accessed July 26, 2026, https://gauravnotes.com/2025/06/15/fitness-functions/
14. Decision Debt: The Invisible Interest Draining Your Team's, accessed July 26, 2026, https://reachmeatrajan.medium.com/decision-debt-the-invisible-interest-draining-your-teams-momentum-and-how-to-pay-it-down-298b1533b700
15. naomijub/transistor: https://crates.io/crates/transistor · GitHub - GitHub, accessed July 26, 2026, https://github.com/naomijub/transistor
16. Structured MADR | Structured MADR, accessed July 26, 2026, https://smadr.dev/