AI System & Agent Design Principles
with API Error Handling Architecture

This document is a concise, language-agnostic reference for building secure, stable, and maintainable AI-driven systems. It extends the core design principles with a standardized API Error Handling Architecture to ensure correctness, auditability, and long-term reliability.

1. Secure by Design

Systems and agents must be built with safety and predictability as defaults.

- Least Privilege: Grant only the minimum required capabilities.
- Explicit Boundaries: Every tool has defined inputs, outputs, and side effects.
- No Implicit Trust: Validate all incoming and outgoing data.
- Auditability: Log tool usage, errors, and state changes.
- Idempotent Actions: Retries must not duplicate or corrupt data.
- Fixed Capability Set: Agents cannot create or extend their own capabilities.

2. SOLID Principles Applied to AI Systems

- Single Responsibility: Each tool or module does one thing.
- Open/Closed: Extend via new components, not modifications.
- Liskov Substitution: Swap implementations without breaking consumers.
- Interface Segregation: Small, task-specific interfaces only.
- Dependency Inversion: Depend on stable contracts, not implementations.

3. DRY (Don't Repeat Yourself)

Avoid duplicating validation, schemas, error handling, or business logic. Centralization increases reliability, security, and maintainability.

4. YAGNI (You Aren’t Gonna Need It)

Do not build speculative capabilities. Add tools only when a real use case exists and keep scopes small and intentional.

5. KISS / Law of Simplicity

Simple systems fail in predictable ways. Keep data flow explicit, reduce hidden branching, and avoid nested orchestration logic.

6. Replaceability & Modularity

LLMs, vector stores, memory layers, tools, and planning modules must be swappable through clear contracts without system-wide refactors.

7. Contracts Everywhere

Use structured, enforced schemas at every boundary to reduce improvisation, standardize behavior, and increase predictability and safety.

8. API Error Handling Architecture (Language-Agnostic)

Error handling is treated as a first-class architectural concern, not ad-hoc logic. Errors are part of the domain and enforced consistently across all system boundaries.

- Domain Truth: Define canonical error codes, categories, and semantics independent of transport, frameworks, or databases.
- Boundary Translation: Map domain errors and unknown failures into transport-specific responses (HTTP, jobs, webhooks) without inventing new meaning.
- Enforced Consistency: All entry points flow through a single choke point. No custom error shapes, no duplicated try/catch logic.

This architecture ensures predictable failures, auditability, and makes AI-generated code operate inside strict guardrails instead of inventing behavior.

9. Practical Checklist

- Does this component do exactly one thing?
- Is there a clear input/output contract?
- Is this capability necessary right now (YAGNI)?
- Does it follow least privilege?
- Is logic duplicated anywhere?
- Can the implementation be swapped safely?
- Does failure remain predictable and auditable?

Core Mindset

Design AI systems as adversarial until proven otherwise. Keep components small, contracts explicit, errors centralized, and behavior predictable. This is how you build AI systems that behave.