Structured Context Management for LLM-Assisted Software Development

Methodology, Design Tradeoffs, and Comparison to Existing Approaches

Author: Garret Sutherland
Affiliation: MirrorEthic LLC
Date: January 2026

Abstract

Large language models used for software development are frequently described as “forgetful,” “hallucination-prone,” or incapable of maintaining understanding across sessions. This paper argues that many of these perceived failures are not intrinsic limitations of the models themselves, but artifacts of stateless interaction under bounded context.

We present a methodology for structured context management that treats the context window as a budgeted working set, governed by attention decay, tiered admission, and co-activation of related artifacts. We describe how this methodology is instantiated in Claude Cognitive, and compare it to other common approaches such as session memory, retrieval-augmented generation (RAG), repository summarization, and manual context attachment.

The goal of this paper is not to promote a specific tool, but to clarify the design space, correct common misconceptions about AI coding failures, and provide a framework for reasoning about context, memory, and coordination in LLM-assisted development.

1. The Misattributed Failure Mode

A common narrative in AI-assisted software development is that models “lose understanding,” “forget decisions,” or hallucinate due to insufficient reasoning ability. In practice, a large class of these failures arises from a simpler cause:

The model is asked to reason without access to the artifacts that define reality for the system in question.

Most AI coding assistants are stateless across sessions and operate under a fixed context window. When relevant files, architectural documentation, or prior decisions are absent, the model must interpolate. The resulting output is often labeled a hallucination, but is more accurately described as a best-effort guess under missing constraints.

Misattributing this failure mode leads developers toward ineffective remedies: anthropomorphic “memory,” over-summarization, or dismissing the tool as unreliable—rather than addressing the interaction design problem.

Illustrative Example

Consider a developer working in a large codebase who asks:

> “How should I implement the authentication flow?”

Without structured context, the model lacks access to existing constraints. It may respond with a generic OAuth or JWT design, invent interfaces, or recommend libraries not used by the project. This output is often described as a hallucination.

With structured context management, the same prompt triggers inclusion of canonical artifacts:

the existing AuthService interface,

documentation describing supported identity providers,

known integration constraints.

The model now references the actual architecture and avoids introducing incompatible patterns. The difference is not a change in model capability, but a change in what the model is allowed to see.

2. Reframing the Problem: Context as a Working Set

From a systems perspective, the context window is not memory in a cognitive sense. It is a finite working set.

The core problem is therefore not recall, but admission control:

which artifacts enter the working set,

at what fidelity,

for how long,

and based on what signals.

Once framed this way, the problem becomes amenable to standard engineering techniques rather than speculative theories of model cognition.

3. Methodology: Design Principles for Structured Context

The following principles guided the design of Claude Cognitive and are broadly applicable regardless of implementation.

3.1 Deterministic Ground Truth Over Inferred Semantics

Many software development questions are binary and factual: does this exist, where is it defined, what constraints apply. For these cases, deterministic artifacts—explicit files and canonical documentation—are more reliable than inferred semantic similarity.

3.2 Budgeted Recall, Not Maximal Recall

Attempting to load all potentially relevant information rapidly exhausts the context window and degrades performance. Instead, context must be explicitly budgeted, with clear ceilings and eviction rules.

3.3 Temporal Relevance as a Control Signal

Relevance is dynamic. Recency provides a natural decay signal that prevents context calcification and ensures the working set reflects current intent rather than historical inertia.

3.4 Co-Activation Reflects Architectural Coupling

Software components are coupled. When one artifact becomes relevant, adjacent artifacts often become relevant as well. Co-activation captures this relationship more reliably than isolated retrieval.

3.5 Separation of Memory Types

Different information serves different purposes and should not be conflated:

Context routing: what the model sees now

Coordination history: what actions have occurred

Human documentation: explanatory material

Separating these reduces ambiguity and failure modes.

4. Claude Cognitive as an Implementation

Claude Cognitive implements the above methodology using two orthogonal mechanisms.

4.1 Attention-Tiered Context Routing

Artifacts are assigned a continuously decaying attention score and placed into one of three tiers:

HOT: fully injected

WARM: truncated injection

COLD: evicted

Promotion occurs through explicit reference or co-activation; decay occurs automatically if not reinforced. This creates a self-regulating working set that respects a fixed context budget.

4.2 Cross-Instance Coordination Pool

In multi-session or multi-developer environments, redundancy arises when instances repeat work already completed elsewhere.

A lightweight, append-only coordination channel allows instances to share high-level state (completed tasks, blockers) without conflating this information with ground-truth documentation. It records what happened, not what is true.

5. Comparison to Other Approaches

Existing tools address adjacent problems but operate in different parts of the design space.

5.1 Session Memory and Recap Systems

Preserve conversational continuity.
Strength: workflow history
Limitation: can perpetuate incorrect assumptions if ground truth is absent

5.2 Retrieval-Augmented Generation (RAG)

Uses embeddings to retrieve semantically similar documents.
Strength: broad semantic recall
Limitations: non-deterministic retrieval, infrastructure complexity, limited auditability

5.3 Repository Summaries and Maps

Provide compressed global structure.
Strength: orientation
Limitation: loss of local detail and dynamic relevance

5.4 Manual Context Attachment

Relies on explicit user selection.
Strength: control
Limitation: high cognitive overhead, inconsistent usage

Claude Cognitive targets working-set admission control, which most approaches do not explicitly address.

6. Enterprise and Government Deployment Considerations

While the methodology is general, it aligns well with regulated environments.

- **Stateless Core Operation**: No hidden model state or opaque memory.

- **Local-Only Context Management**:
  Configuration and state are expressed entirely through a project-local filesystem structure (the .claude/ directory), rather than IDE-specific plugins or editor-dependent state. This ensures consistent behavior across terminal workflows, VS Code, and other editors.

- **Auditability and Governance**:
  Because configuration is filesystem-based, context behavior can be reviewed, diffed, and governed using standard version control processes, supporting security review and regulated deployment environments.

- **Deterministic Admission**: Explicit rules govern what enters context and why.

- **Tool Independence**: Documentation artifacts are model-agnostic.

These properties simplify security review and support air-gapped or high-assurance deployments.

7. Limitations and Failure Modes

This methodology does not eliminate all risk:

keyword routing requires curation

documentation can become stale

co-activation can over-pull context

coordination logs require disciplined use

These are explicit tradeoffs and can be mitigated through tooling and process.

8. Implications for Tooling and Practice

Correctly attributing AI failures to interaction design rather than cognition has downstream effects:

Tooling: Context admission logic should be visible and inspectable.

Documentation: Becomes machine-consumable, not merely human-readable.

Developer Mental Models: Reduced anthropomorphism, improved judgment.

Governance: Explicit context limits reduce unintended inference.

Better context management improves outputs—and improves how engineers reason about AI systems themselves.

9. Conclusion

AI coding assistants do not primarily fail because they “forget.” They fail because we ask them to reason without supplying the artifacts that define reality.

By treating context as a managed working set rather than mystical memory, hallucinations and redundancy can be dramatically reduced without changing the underlying model.

Claude Cognitive demonstrates one concrete implementation. The broader contribution is the methodology.