Context Engineering

Augmented Software Engineering

Edwin Van Dillen · CTO

October 2026

TODAY'S AGENDA

What We'll Cover

01
The Context Engineering Shift
From crafting prompts to curating information

02
The Context Window: Advanced
From working memory to managed resource

03
Context Artifacts
AGENTS.md · SPEC.md · Caching

04
Dynamic Context: Retrieval
RAG · Tool Use · MCP · Agentic Retrieval

05
Context in the Agent Loop
Lifecycle · Handoff · Compression · Anti-Patterns

06
Key Takeaways
What we covered and your next steps

AI-Aware Engineering Series

Your Learning Path

SOGYO

01
FOUNDATION

Prompting

Mental models, RCTF framework, prompt optimisation & meta-prompting

02
INFORMATION ARCHITECTURE

Context Engineering

Context layers, token budgets, AGENTS.md, RAG & MCP

← YOU ARE HERE →

03
AUTONOMOUS SYSTEMS

Agents

Agent loop, agentic patterns, Copilot Agents, MCP, safety & evaluation

01

The Context Engineering Shift

From crafting prompts to curating information

What is Context Engineering?

Prompt engineering is what you say. Context engineering is what the model knows.

WHAT IT IS

The discipline of curating what information the model has at each step

Deciding what goes in, what stays out, and when

Architecture of information — not just phrasing

The key bottleneck at L3 and L4

Context shape determines output quality

Design your context

WHAT IT IS NOT

Just writing a better prompt

Dumping your entire codebase into the chat

Only about the system prompt

A one-time task — it is ongoing

Prompt shape determines output style

Not a one-off fix

Prompt Engineering vs. Context Engineering

Both improve AI output — but they operate at different levels.

PROMPT ENGINEERING

Focuses on the words you use

One interaction at a time

You control the phrasing

Skill: knowing what to ask

Result: better individual responses

Prompt quality

CONTEXT ENGINEERING

Focuses on the information you provide

Designed across the full task lifecycle

You control the information architecture

Skill: knowing what the model needs to know

Result: better entire workflows

Context quality

The Information Gap

Most AI failures are not model failures — they are context failures.

SIGNS OF CONTEXT FAILURE

- Agent implements the wrong pattern
- Agent ignores your existing code
- Agent repeats your naming mistakes
- Agent hallucinates an API method
- Agent drifts after 30 messages

Context failure looks like model failure

ROOT CAUSE

→ Didn't know your team uses Clean Architecture

→ Couldn't see the relevant file

→ No conventions were in context

→ Had no live documentation

→ Context window is full and polluted

Every one is a context design problem

02

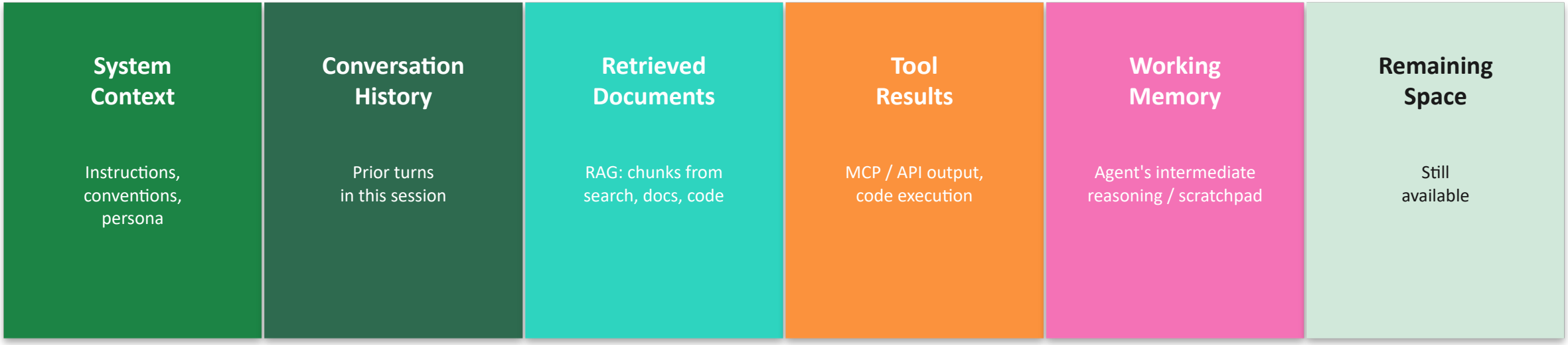
The Context Window: Advanced

From 'working memory' to managed resource

You saw this in Prompting V1.4 — now we go deeper

Full Anatomy of the Context Window

Everything the AI can see at once — now with 6 components. The last two are where context engineering happens.



New in this deck: Retrieved Documents · Working Memory — these are where context engineering happens

Context Pollution: More Is Not Better

A full context window is not the same as a good context window.

HEALTHY CONTEXT

- Precise system prompt with team conventions
- Only the relevant files — not the entire repo
- Retrieved docs scoped to this specific task
- Clear task definition without unrelated history

→ Model reasons with signal

High signal-to-noise ratio

POLLUTED CONTEXT

- 30 messages of debugging detours still open
- Entire codebase pasted in 'just in case'
- Contradictory instructions from two sessions
- Tool results from failed branches, never cleaned

→ Model reasons with noise

Context debt accumulates

Context Budget Management

Not all context is worth the tokens. Decide deliberately.

ALWAYS IN

- Team conventions & architecture rules
- Current task specification
- Relevant interface definitions
- Active working plan

RETRIEVED ON DEMAND

- Specific function implementation
- Live ticket or PR data
- Documentation for a dependency
- Error logs for failing tests

NEVER IN

- Unrelated conversation history
- Entire files when a function suffices
- Duplicate or superseded instructions
- Output from cancelled task branches

Context Has a Cost

Every token in your context window costs money. Context engineering is also cost engineering.

WHAT TOKENS COST (per 1M, indicative — prices fall yearly)

Model	Input	Cached input
Provider A, balanced	~$3	~$0.30 (90% off)
Provider B, balanced	~$2.50	~$1.25 (50% off)
Provider C, balanced	~$1.25	~$0.31 (75% off)

1,000 tokens ≈ 750 words ≈ 50 lines of C#

WHAT THAT MEANS IN PRACTICE

System prompt 5K tokens × 1,000 calls/day = 5M tokens/day

Without caching: ~$15/day → $450/month just for context overhead

With caching: ~$1.50/day → $45/month (10× cheaper)

Sending 50K irrelevant tokens per call: +$150/day extra

→ Caching and scoping can cut context cost by up to ~90%

Context engineering is not just about quality — it's about cost.

SOGYO

03

Context Artifacts

Persistent context that works across every session

AGENTS.md · copilot-instructions.md · .agent.md · SPEC.md

The Context Artifact Stack

Most teams only engineer at Task Level. The leverage is at Repo and Org Level.

ORG LEVEL	Team conventions · Architecture decisions · ADRs	*Changes rarely*
REPO LEVEL	AGENTS.md · copilot-instructions.md · .agent.md	*Changes per sprint*
SESSION LEVEL	Working plan · Retrieved docs · Tool results	*Changes per session*
TASK LEVEL	Issue description · PR brief · Ad-hoc prompt	*Changes per task*

Most teams live only at Task Level — the biggest leverage is at Repo and Org Level

CONTEXT ARTIFACTS · ORG LEVEL

GitHub Copilot Tiers: What Context Levels Are Available

Your subscription tier determines how high up the context stack you can engineer

Feature	Individual (single developer)	Business Organisations	Enterprise Large / multi-org
copilot-instructions.md *Repo-level persistent context*	✓	✓	✓
Path-specific .instructions.md *applyTo per folder / file type*	✓	✓	✓
Org-level instructions *Admin-configured, all repos in org*	✗	✓	✓
Multiple organisations *Separate orgs per client / team*	✗	✗	✓
Per-team request budgets *Premium request quota per sub-org*	✗	✗	✓
Policy management & audit *Compliance controls, usage logs*	✗	✓	✓

Multi-client shops: org-level instructions need Business+, separate client orgs need Enterprise. Plans change often.

copilot-instructions.md as Context Engineering

Not configuration — your team's persistent context layer. AGENTS.md is the cross-tool version.

.github/copilot-instructions.md

```
# Project: PaymentService API

## Tech Stack
- .NET 8 · ASP.NET Core · EF Core 8
- Clean Architecture (Domain/App/Infra)
- MediatR · FluentValidation · xUnit

## Conventions
- Commands + Handlers in Application layer
- No static classes in Domain
- All endpoints return ProblemDetails on error

## What NOT to do
- No try/catch in Domain logic
- No service locator pattern
- Never suggest jQuery or .NET Framework
```

WHAT THE MODEL READS

Tech Stack → model picks the right patterns and libraries

Conventions → model enforces architecture without being told

What NOT to do → eliminates the most common wrong suggestions

This file = your team's persistent context layer

Path-Specific Instructions: Load the Right Context for the Right File

applyTo frontmatter makes context conditional — only loaded when the model touches matching files. Keeps the context window smaller and more precise.

The problem

One copilot-instructions.md with C#, TypeScript and Gherkin rules combined.

Every rule loads on every file — extra tokens and noise.

The solution: applyTo

Split into .instructions.md files with an applyTo glob.

Copilot loads a file only when the edited file matches.

frontend.instructions.md
TypeScript + React

```
---
applyTo: "frontend/src/**"
---
# Frontend Standards
## Styling
Use Tailwind utility classes.
## State Management
Zustand for global state.
```

Loads when: frontend/src/**

csharp.instructions.md
C# / .NET

```
---
applyTo: "**/*.{cs,csproj}"
---
# C# Conventions
Use CSharpier as formatter.
Run: dotnet csharpier format .
No manual style overrides.
```

Loads when: **/*.{cs,csproj}

gherkin.instructions.md
Gherkin / BDD

```
---
applyTo: "**/*.feature"
---
# Feature Files
Copy UC Gherkin verbatim.
Never paraphrase steps.
Ambiguous? Stop and ask.
```

Loads when: **/*.feature

copilot-instructions.md
Always-on, repo-wide

.instructions.md
Loaded via applyTo

.agent.md
Agent persona

.prompt.md
Specific task or flow

SKILL.md
Skill folder, on demand

The Spec as Context

A bad issue gives bad code. A spec gives good code.

ISSUE (LOW CONTEXT)

"Add JWT authentication"

Agent makes all the design decisions

Output surprises you

You spend time correcting

You are still the driver

L2 result

SPEC (HIGH CONTEXT)

Goal + Scope + Constraints + Acceptance Criteria

Agent executes your decisions

Output matches your mental model

You spend time reviewing

You are the PM

L4 result

SPEC.md: Your Best Context Investment

Copy and adapt. Five minutes of writing → significantly better output.

```
# SPEC: [Feature Name]

## Goal
One sentence: what should exist that doesn't exist now?

## Context
- Relevant existing code: [file/class references]
- Depends on: [service/API/database]
- Must NOT break: [existing behavior]

## Constraints
- Architecture: [layer rules]
- Style: [naming / patterns to follow]
- Out of scope: [explicitly excluded]

## Acceptance Criteria
- [ ] [Testable condition 1]
- [ ] [Testable condition 2]
- [ ] All existing tests still pass
```

A good spec is context engineering — it tells the agent everything it needs, and nothing it doesn't.

Context Caching: Engineer for Cache Hits

The order of your context determines whether it is cached — and that determines your cost and latency.

HOW CACHING WORKS

- Providers cache the prefix of your context (the beginning)
- Same prefix across calls → cache hit → 50–90% cost & latency reduction
- Cache invalidates when the prefix changes
- TTL: minutes to an hour or more (varies by provider)
- Only the stable part is cached — dynamic parts never are

HOW TO ARCHITECT FOR CACHE HITS

← CACHED (stable, large)

System prompt · Team conventions · Large specs · Few-shot examples

← NOT CACHED (changes/call)

Retrieved docs · Conversation history · Current user message

Put your largest, most stable content first. The more it doesn't change, the more you save.

Context Artifact Checklist

What goes where — and when to update it.

Artifact	Belongs in	Updated when
Team conventions, architecture rules	AGENTS.md / copilot-instructions.md	Per sprint / ADR change
Agent persona, task focus, output format	.agent.md	Per agent role
Feature goal, constraints, acceptance criteria	SPEC.md	Per feature
Architectural decisions & rationale	ADR.md	Per significant decision
Active task plan, intermediate steps	Working memory (session)	Per session
Live data: tickets, PRs, logs	Retrieved via MCP / tools	Real-time

04

Dynamic Context: Retrieval

Give the agent exactly what it needs, exactly when it needs it

RAG · Tool Use · MCP · Agentic Retrieval

Static vs. Dynamic Context

Both are essential — they serve different purposes.

STATIC CONTEXT

Written once, always in the window

Fixed at session start

Gets stale over time

Limited by token budget

Best for: conventions, architecture, standards

Defines how the agent thinks

DYNAMIC CONTEXT

Retrieved at the moment it is needed

Fetched fresh from the source

Always current

Scales beyond token limits

Best for: live data, implementations, tickets, logs

Gives the agent what it needs to act

RAG: Retrieval-Augmented Generation

RAG gives the agent access to knowledge it couldn't fit in context upfront.

THE PATTERN

1. Agent receives task
2. Identifies what information it needs
3. Searches knowledge base (semantic search)
4. Retrieves the relevant chunks
5. Injects them into context
6. Reasons with live, scoped information

Retrieve what matters

IN PRACTICE (Copilot / Claude Code)

Agent needs to implement a Repository pattern

Searches codebase for existing Repository code

Retrieves IOrderRepository interface + one example

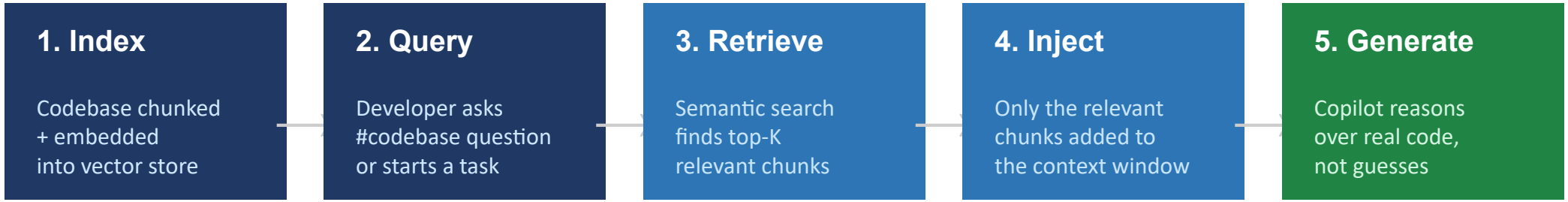
Generates new repo that matches exact existing patterns

No hallucinated method names — it saw the real interface

Real code, real patterns

Copilot Codebase Indexing: RAG Applied to Your Repository

VS Code indexes your codebase once. Every codebase search (#codebase, or automatically in agent mode) retrieves only the relevant fragments. This is RAG, running in the background.



Without codebase indexing

Copilot sees only the currently open file + manually @-referenced files. You must explicitly add context. The model cannot reason across the repo.

VS

With codebase indexing (VS Code)

Codebase searches trigger semantic search. The most relevant chunks are retrieved and injected automatically. The context window stays small — only what matters is included.

Codebase indexing = RAG for your repo: every codebase search retrieves only the code that matters.

Agentic RAG: The Agent Decides What to Retrieve

Standard RAG is a fixed pipeline. Agentic RAG is driven by the agent's own reasoning.

	STANDARD RAG	AGENTIC RAG
Retrieval timing	Fixed pipeline step before reasoning	Agent decides when during reasoning
Rounds	One retrieval per query	Multiple rounds as the task evolves
Who defines scope	You define what is searched	Agent identifies gaps and fills them
Context update	Static injection before the call	Context evolves with the task

An agent that retrieves is an agent that learns as it works

Tool Results as Context

Tools don't just perform actions — they inject live context.

WITHOUT TOOL CONTEXT

Task: "Fix the auth bug from ticket #4521"

Agent guesses what the bug description says

Agent guesses which code is relevant

Agent writes a generic fix

Agent may introduce regressions

Working blind

WITH TOOL CONTEXT (MCP)

Task: "Fix the auth bug from ticket #4521"

Agent calls Jira → reads bug description + repro steps

Agent calls GitHub → reads PR that introduced the bug

Agent reads Sentry logs → sees exact stack trace

Agent writes targeted, informed fix

MCP = context delivery infrastructure

SOGYO

05

Context in the Agent Loop

How context evolves as the agent works

Context lifecycle · Handoff · Compression · Anti-Patterns

Context Lifecycle During a Task

At each step the agent both consumes context and produces new context for the next step.

Loop Step	Context IN	Context OUT
PLAN	Spec · conventions · codebase overview	Task plan · identified files
ACT	Specific file contents · tool results	Code changes · intermediate decisions
OBSERVE	Test results · compiler output · logs	Assessment · error classification
ADAPT	Delta: what changed vs. what failed	Revised plan or next step

Context Handoff in Multi-Agent Systems

Agent B starts with a blank slate unless Agent A writes a handoff first.

handoff.md — structured context transfer

```
# Handoff: Authentication Implementation

## What was done
- Added JwtService to Application layer
- Interface: IAuthService with 2 methods
  (issue, validate)

## Key decisions
- Used RS256 (asymmetric) — ADR-007
- Token TTL: 15 min (agreed with team)

## What's next
- Implement AuthController endpoints
- Wire up DI in Program.cs
- Do NOT change the IAuthService interface
```

THE SOLUTION

Agent A writes handoff before completing

Includes: decisions made + rationale + constraints

Agent B starts with this as first context

Continuous context — not a fresh start

Each agent knows what the other decided and why

Context Compression: Managing Long Tasks

Long sessions fill the window. Most agent tools now compact automatically — know what they keep and what they drop.

Strategy	When to use	How
Rolling summary	Long debugging sessions	Agent summarises last N turns before they leave the window
Decision log	Multi-step planning tasks	Keep only the decisions — not the reasoning that produced them
Fresh session + handoff	Task spans hours or days	End session with structured handoff; start new session with it
Selective retention	Tool-heavy workflows	Keep successful tool results; evict failed attempt details

Context Anti-Patterns: What Goes Wrong

Every anti-pattern is a context architecture decision made by default instead of by design.

Context Dumping

Paste the entire codebase 'just in case'

→ Model spends attention on irrelevant code and misses what matters

Session Sprawl

One conversation thread for 10 different tasks

→ Earlier context contradicts later context — model drifts

Instruction Conflict

System prompt says X, user prompt says Y

→ Unpredictable behaviour — model picks one interpretation arbitrarily

Stale Context

Spec was updated but context wasn't refreshed

→ Agent builds against the old requirements

06

Key Takeaways

What we covered and your next steps

The Three Questions

Ask these before every significant AI task.

01

What does the model know?

→ Audit your context: system prompt, session history, loaded files

02

What does it need to know?

→ Map information requirements to the task before starting

03

What's getting in the way?

→ Find the noise: old turns, irrelevant files, conflicting instructions

KEY TAKEAWAYS · CHECKLIST

Context Engineering Checklist

Two horizons. Start with what you can do alone — today.

DO THIS NOW

- ☐ Does your repo have an AGENTS.md (or copilot-instructions.md) with stack + conventions?
- ☐ Do you write a 5-line spec before long agent tasks?
- ☐ Do you start a fresh session for new, unrelated tasks?
- ☐ Do you paste only the relevant function — not the whole file?

NEXT SPRINT

- ☐ Set up MCP for Jira/GitHub so agents have live issue context
- ☐ Create a SPEC.md template for your team
- ☐ Define your context handoff format for multi-step workflows
- ☐ Review your AGENTS.md with the whole team

The Series Continues

Prompting V1.4	How to speak to AI: RCTF, reasoning models, review, meta-prompting
Agentic Systems V2.3	How AI acts: 6 patterns, MCP, guardrails, infrastructure
Context Engineering V1.5	How to feed AI: artifacts, retrieval, lifecycle

Further exploration:

→ Evaluation — how do you measure context quality?

→ RAG in production — vector databases, chunking, hybrid search

→ Specification Engineering — from SPEC.md to executable specs

KEY TERMS

Concepts introduced in Prompting — referenced throughout the series

5 Levels of AI Coding (L0 – L5)	Dan Shapiro's maturity framework: L0 = spicy autocomplete, L1 = coding intern, L2 = pair programming, L3 = you manage the AI, L4 = you're the PM, L5 = dark factory.
RCTF Framework	Role · Context · Task · Format. The four-element prompt structure from Prompting. Context and intent carry the most weight; the role the least.
Team of Five	Brice Challamel's mental model: you at the centre, plus four AI roles — Assistant, Coach, Expert and Creative Partner. You direct the team; the AI fills the roles.
Intent First	Lead with why before what. From Prompting: the goal and acceptance criteria matter more than the persona — for prompts, specs and agents alike.
Context Window	The fixed-size buffer of tokens the model can see at once. Introduced in Prompting; deep-dived in Context Engineering. Everything the model knows about a task lives here.



Capstone Exercise

Build a PR Review Agent — Context Engineering phase

CAPSTONE EXERCISE

Build a PR Review Agent — Design the Context File Set

Objective: Design the context layer that gives the PR review agent the right information at the right time.

Scenario: You have written the RCTF prompt in Prompting phase. Now design the file-based context architecture that makes it repeatable and maintainable across all repositories in your organisation.

Your tasks

AGENTS.md	Write the team-level instructions (AGENTS.md or copilot-instructions.md): coding standards, PR checklist, severity, escalation.
SPEC.md	Define the agent's scope: what it should and should not review, known exceptions, and tech stack specifics.
Context layers	Map information to context layers (persistent / session / task). What goes in a file, what in RAG?
Dynamic context	Identify what to retrieve at runtime: changed files, related issues, test results. Design the retrieval flow.
Token budget	Estimate the token cost per review and identify what to cut if the context window gets tight.

Deliverable: AGENTS.md + SPEC.md + a context map of where information lives.