EACOMM CORPORATION

AI-Assisted Software Development Lifecycle (SDLC)

Research, Evaluation, Deployment, and Intelligent Platform Development

---

BY:

VINCENT B. ACHA

REYMEL R. MISLANG

HARLYN P. NEBREJA

DARYLL C. TUPAZ

JENILYN W. ZAULDA

July 2026

Table Of Contents

11th Floor CyberOne., Eastwood City Cyberpark, Bagumbayan, Quezon City, Philippines 1110
Tel. (632) 7217 0372 Fax. (632) 89126745 E-mail: info@eacomm.com Web: www.eacomm.com

Executive Summary

This report addresses the three-week scope of work for evaluating and deploying AI-assisted tools across the full software development lifecycle (SDLC): planning, architecture/design, development, documentation, testing/QA, and maintenance/observability. It also proposes a design for a Retrieval-Augmented Generation (RAG) knowledgebase system that can be integrated across these tools.

The research finds that no single AI coding agent, model, or platform "wins" every SDLC stage. The strongest results come from a small, deliberately-chosen stack — one open-source, model-agnostic coding agent; one or two frontier LLMs matched to task type; a lightweight AI-aware project management tool; an AI-assisted wireframing tool; a docs-as-code pipeline; an AI-native test generation layer; and an LLM observability tool — wired together with a shared RAG knowledgebase so that every tool answers from the same source of truth.

Headline Recommendation

Adopt Cline (open-source, Apache-2.0, BYOK, model-agnostic) as the core coding agent, paired with Claude for architecture/reasoning/documentation-quality work and a cheaper high-throughput model (e.g. Gemini or a routed model) for high-volume/boilerplate tasks. Pair this with Plane (self-hosted, AI-native, open-core) for planning, a docs-as-code Markdown/GitBook-style pipeline for documentation, an AI-native open-source testing layer (Playwright/Cypress plus an AI test-generation tool) for QA, and Langfuse for LLM/RAG observability.

Week 1: Research & Discovery

Week 1 covers three deliverables from the scope of work: (1) a survey of AI tools across every stage of the SDLC — not just coding; (2) an evaluation of which AI models perform best for which SDLC use case; and (3) research into RAG implementation and a proposed design for a simple knowledgebase system.

1.1 AI Tools Across the SDLC

The table below summarizes the tool landscape researched for each SDLC phase, with emphasis on open-source and BYOK (bring-your-own-key) options as requested in the scope of work.

A. Planning & Project Management

Traditional PM tools are adding AI copilots rather than being replaced outright. For a small team wanting self-hosted, model-agnostic AI features, three tools stand out:

11th Floor CyberOne., Eastwood City Cyberpark, Bagumbayan, Quezon City, Philippines 1110
Tel. (632) 7217 0372 Fax. (632) 89126745 E-mail: info@eacomm.com Web: www.eacomm.com

- **Plane** — open-core (AGPL-3.0 Community Edition), positions itself as an AI-native Jira/Linear alternative. Ships natural-language workspace queries, AI agents that can be assigned to work items, automation ("Build mode"), and a native MCP server. On self-hosted instances it runs BYOK, so no workspace data leaves your infrastructure. Strongest fit for a small dev team that wants Jira-like structure without Jira's cost or lock-in.
- **OpenProject** — mature, GPLv3, strong for classic/agile/hybrid workflows and regulated environments; AI features (status-report generation, Ask AI, summarization) are newer and more supportive/decision-assist in nature rather than agentic.
- **Taiga / Leantime / Redmine** — lighter-weight open-source alternatives; Leantime is notable for baking in research/hypothesis-driven planning boards, useful for a project like this one that mixes research and delivery.

Recommendation for this project: Plane, self-hosted, for its native MCP server (lets it be wired into the same agent stack as the coding tools) and BYOK AI.

B. System Architecture, Design & Wireframing

This is the most fragmented category — there is no single open-source leader analogous to Cline in coding. Options split into whiteboarding, AI wireframe generators, and diagram/architecture tools:

- **Penpot** — the strongest open-source (MPL-2.0) option for UI/UX design and wireframing, with developer-friendly handoff; lacks the natural-language "generate a screen from a prompt" capability of newer AI-first tools.
- **Pencil Project / Akira** — lightweight open-source wireframing/sketch tools, no AI generation, useful as a zero-cost fallback.
- **AI-first commercial tools** (UXMagic, Visily, Uizard, Magic Patterns, UX Pilot) — generate wireframes or production-ready UI from a text prompt or screenshot; none are open-source, but several (Magic Patterns, UX Pilot) integrate directly with Claude/OpenAI APIs, so they can be paired with a BYOK key rather than a locked-in model.
- **Diagramming for system architecture** — Mermaid (open-source, text-to-diagram, renders natively in Markdown/GitHub) is the most practical BYOK-friendly option for architecture diagrams, sequence diagrams, and ER diagrams generated directly by an LLM.

Recommendation: use Mermaid (via the coding agent/LLM) for system/architecture diagrams since it is plain text, versionable in Git, and any LLM can generate it; use Penpot for actual UI wireframes if a dedicated design tool is needed, or a BYOK-based generator like Magic Patterns if speed-to-prototype matters more than open-source purity.

11th Floor CyberOne., Eastwood City Cyberpark, Bagumbayan, Quezon City, Philippines 1110
Tel. (632) 7217 0372 Fax. (632) 89126745 E-mail: info@eacomm.com Web: www.eacomm.com

C. Development & Implementation (Coding)

This is the most mature category. The open-source, model-agnostic coding agent landscape as of mid-2026:

- **Cline** — the tool named in the scope of work. VS Code extension plus a maturing standalone CLI/SDK. Apache-2.0, ~58–62k GitHub stars, 5M+ installs. Plan/Act workflow requires human approval at every file write, terminal command, or browser action — the most auditable option, which matters for a team wanting to review AI output. Model-agnostic (30+ providers, local models via Ollama/LM Studio). Supports MCP for external tool integration.
- **OpenHands** — MIT/Apache, ~65–75k stars, the strongest "autonomous agent platform" option: sandboxed Docker runtime, headless mode for CI/CD, resumable sessions. Best when you want an agent to work a ticket end-to-end with less step-by-step approval.
- **OpenCode** — the most-starred open-source coding agent (~165k stars), terminal-native, provider-agnostic, positioned as the open-source alternative to Claude Code.
- **Aider** — git-native terminal pair programmer; every edit becomes a commit, which gives a very clean audit trail for a small team documenting its process.
- **Kilo Code / Roo Code** — Cline-family forks. Note: Roo Code was archived/shut down in May 2026, so Kilo Code is the recommended successor if a multi-mode (Plan/Architect/Debug) workflow beyond Cline's is wanted.

> **Note on the ecosystem**
>
> The coding-agent space changed quickly during this research window: Roo Code was archived in May 2026, Google is retiring the free-serving Gemini CLI, and Continue pivoted from an IDE assistant to a CI code-review product. Any tool selected in Week 2 should be re-verified against its current repository status before installation, since this category moves on a roughly monthly cadence.

D. Documentation (User/Admin Manuals)

Documentation tooling splits into two useful patterns for this project:

- **Docs-as-code with an AI layer** — GitBook and Mintlify both keep documentation in Git, support Markdown, and add AI features (drafting, Q&A search, auto-maintenance) on top. Mintlify in particular is built around API-documentation with an AI assistant baked in.
- **Code-to-docs generation** — tools that connect to a Git repository and auto-generate API references, UML/sequence diagrams, and README content on every commit, keeping docs in sync with the codebase

11th Floor CyberOne., Eastwood City Cyberpark, Bagumbayan, Quezon City, Philippines 1110
Tel. (632) 7217 0372 Fax. (632) 89126745 E-mail: info@eacomm.com Web: www.eacomm.com

automatically. This is the most relevant pattern for the "document any custom source code/apps developed" requirement in Week 3.

- **General LLM drafting** — for the actual user/admin manuals (prose, not API refs), a general-purpose model (Claude or GPT) used with strong source inputs (specs, tickets, screen recordings) can get a manual roughly 80% of the way, per technical-writing practitioners, with a human editing pass for accuracy and product-specific nuance.

Recommendation: keep manuals as Markdown in the same Git repo as the code (docs-as-code), draft first passes with an LLM fed the actual source code and configs, and use a lightweight tool (Mintlify or plain GitBook) for the developer-facing manual and a plain Word/PDF export for any end-user-facing manual.

E. Testing & QA

AI testing tools fall into three tiers, useful to understand before picking one:

- **AI-bolted-on** — classic frameworks (Selenium, Playwright, Cypress, Appium, Robot Framework) with AI plugins for self-healing selectors. The AI patches tests; it doesn't author them.
- **AI-assisted scripting** — tools like Stagehand or Shortest that let an LLM help write Playwright/Cypress-style scripts from natural language.
- **AI-native generation** — the LLM agent is the primary author: it reads product intent, decides what to test, writes and executes the steps, and updates tests when the UI changes. This tier includes both open-source options (EvoMaster for API fuzzing, Magnitude) and commercial platforms.

Recommendation: Playwright as the execution framework (open-source, well-documented, broad browser support) paired with an AI-assisted layer for generating the initial test scripts from natural-language descriptions of the end-to-end pilot project built in Week 3. For a project this size, full AI-native autonomous QA platforms are likely more than is needed — the value is in faster authoring, not unattended execution.

F. Maintenance & Observability

Because this project's own deliverable is an AI-powered system (the RAG knowledgebase plus any custom tooling), observability needs to cover both LLM behavior and normal application health:

- **Langfuse** — open-source (MIT core), self-hostable via Docker, captures traces/sessions/prompt versions across LangChain, LlamaIndex, the OpenAI SDK, and other frameworks. The most natural fit for a small team that wants to self-host and inspect RAG retrieval traces.

11th Floor CyberOne., Eastwood City Cyberpark, Bagumbayan, Quezon City, Philippines 1110
Tel. (632) 7217 0372 Fax. (632) 89126745 E-mail: info@eacomm.com Web: www.eacomm.com

- **OpenObserve** — open-source (AGPL-3.0), unifies LLM traces with general infrastructure logs/metrics/traces in one deployment — useful if the team doesn't want a separate DevOps monitoring stack alongside the AI one.
- **Arize Phoenix** — open-source, specifically strong for RAG pipeline debugging (retrieval quality, hallucination scoring).

Recommendation: Langfuse for LLM/agent/RAG-specific tracing (open-source, self-hosted, directly relevant to "maintenance/observability" in the scope), since it directly answers "why did the RAG system retrieve this?" and "did this agent step fail?" — the two observability questions most relevant to this project.

1.2 AI Model Evaluation by SDLC Use Case

The scope of work asks the team to "play around" with and document how OpenAI, Google, Anthropic, and open-source models perform for each SDLC use case. Based on current (mid-2026) public benchmarks and practitioner comparisons, no model dominates every category — the honest finding across nearly every independent comparison is that teams get the best results by routing tasks to the model that is strongest for that specific job, rather than standardizing on one vendor.

SDLC Use Case	Strongest Fit	Why	Also Consider
Whole-repo / complex multi-file coding	Claude (Opus tier)	Leads real-world bug-fixing benchmarks (SWE-bench Verified) and multi-file logic reasoning; large context + high output length suit big refactors.	GPT (Codex variant) for terminal/CI automation; Gemini for very large codebases needing 1M-token context.
Algorithmic / competitive coding problems	Gemini (Pro tier)	Leads on algorithmic-reasoning benchmarks (e.g. LiveCodeBench) that reward pure logic over real-world bug-fixing.	Claude and GPT are close behind; differences are workload-dependent.
Terminal / DevOps / CI automation	GPT (Codex CLI variant)	Consistently tops terminal-automation and shell-scripting benchmarks; purpose-built for agentic CLI workflows.	Cline/OpenHands as the harness, any model as the backend.
Documentation & user manuals (prose quality)	Claude	Practitioner and benchmark consensus rates Claude's prose as the most natural and least "AI-sounding" for long-form writing; strong at producing flowing, readable manuals.	GPT for highly technical, precision-heavy reference docs.
Architecture / system	Claude or Gemini	Both lead graduate-level	Use whichever

11th Floor CyberOne., Eastwood City Cyberpark, Bagumbayan, Quezon City, Philippines 1110
Tel. (632) 7217 0372 Fax. (632) 89126745 E-mail: info@eacomm.com Web: www.eacomm.com

SDLC Use Case	Strongest Fit	Why	Also Consider
design reasoning		reasoning benchmarks (GPQA Diamond) relevant to weighing architectural trade-offs; scores are within noise of each other.	the team already has access to — the gap is marginal.
High-volume / low-complexity tasks (formatting, boilerplate, simple test generation)	Cheapest capable model (e.g. Gemini Flash, Claude Haiku, GPT mini)	60–80% of routine coding-agent requests are simple enough that a frontier model is wasted spend; routing these to a small/cheap model cuts cost without hurting quality.	Local open-weight models via Ollama if data cannot leave the network.
Multimodal (screenshots → wireframes, image-based bug reports)	Gemini	Widest native multimodal support (text, image, audio, video) and the largest context window for feeding in screenshots plus specs together.	Claude and GPT both handle images competently for simpler cases.

For the RAG-specific use case (Week 1's third research item), the embedding/generation split matters more than the "which frontier model wins" question: retrieval quality is driven by the embedding model and chunking strategy, not by which LLM ultimately writes the answer. Any of the three vendors' embedding models are adequate; the generation step (turning retrieved chunks into an answer) is where model choice in the table above applies.

Practical takeaway for Week 2

Rather than adopting a single "official" model, configure the coding agent (Cline) with BYOK access to at least two models: one frontier reasoning/writing model (Claude) for architecture, code review, and documentation, and one fast/cheap model for high-volume boilerplate. This mirrors what practitioner teams report doing in 2026 and avoids over-spending frontier-model budget on routine work.

1.3 RAG Implementation Research & Knowledgebase System Design

Retrieval-Augmented Generation grounds an LLM's answers in a specific, private set of documents instead of relying only on what the model memorized during training. This matters directly for this project because the knowledgebase is meant to hold the team's own research notes, tool evaluations, source code, and manuals — content no public model has seen.

How RAG works (pipeline)

- **Ingest** — pull in source documents: PDFs, Markdown/Confluence pages, code comments, tickets, meeting notes.

11th Floor CyberOne., Eastwood City Cyberpark, Bagumbayan, Quezon City, Philippines 1110
Tel. (632) 7217 0372 Fax. (632) 89126745 E-mail: info@eacomm.com Web: www.eacomm.com

- **Clean & chunk** — strip boilerplate/formatting and split documents into semantically coherent pieces. Current practitioner guidance (2026) suggests starting around 300–500 tokens per chunk with 10–15% overlap, then tuning from there.
- **Embed** — convert each chunk into a vector (numerical representation of meaning) using an embedding model.
- **Store & index** — save the vectors plus metadata (source document, section heading, page/line number, parent-chunk ID) in a vector database, so results can be filtered and cited back to their source.
- **Retrieve** — at query time, embed the user's question and fetch the most similar chunks; production guidance increasingly recommends hybrid search (vector similarity plus keyword/exact-match) because pure vector search misses exact identifiers like product codes or function names.
- **Generate** — pass the retrieved chunks plus the original question to an LLM, which writes an answer grounded in that context rather than from memory.

Proposed Knowledgebase System Design for this Project

A simple, self-hostable design that fits a 3-week timeline and can plug into the tools chosen in section 1.1:

Layer	Recommended Choice	Rationale
Vector database	Chroma or Qdrant (self-hosted)	Both open-source, quick to stand up locally, no cloud dependency for a proof-of-concept knowledgebase.
Embedding model	BYOK embedding endpoint (OpenAI, Google, or Voyage/Anthropic-compatible)	Keeps the whole stack model-agnostic and consistent with the "bring your own API key" requirement.
Orchestration	LlamaIndex or LangChain	Both are the de facto open-source frameworks for building the ingest → chunk → embed → retrieve pipeline without writing it from scratch.
Retrieval strategy	Hybrid search (vector + keyword) with metadata filtering	Naive vector-only retrieval fails on exact-match lookups (tool names, error codes, file paths) which this project's content will contain heavily.
Generation model	Claude (or whichever model the team standardizes on per section 1.2)	Strong grounded-answer quality and natural prose for a knowledgebase that will be read by humans, not just machines.
Observability	Langfuse	Traces exactly which chunks were retrieved for a given answer — essential for debugging "why did it say that" during the Week 3 pilot.
Content sources for v1	Project research notes, tool evaluation docs, any custom	Matches what the team will actually have produced by the time the knowledgebase

11th Floor CyberOne., Eastwood City Cyberpark, Bagumbayan, Quezon City, Philippines 1110
Tel. (632) 7217 0372 Fax. (632) 89126745 E-mail: info@eacomm.com Web: www.eacomm.com

Layer	Recommended Choice	Rationale
	source code + comments, manuals produced in Week 3	needs content.

This design deliberately avoids fine-tuning a model: 2026 practitioner consensus (and the original RAG research) is that RAG delivers most of the benefit of "teaching" a model new, private knowledge at a fraction of the cost and complexity of fine-tuning, and it keeps the knowledgebase content easy to update — add or edit a document and re-index, rather than retraining anything.

Week 2: Deploy and Develop

Week 2's scope is: agree on the team's tool/model stack, install and deploy what's freely available, and build the RAG knowledgebase, exploring how it integrates with the tools selected.

2.1 Proposed Stack for Team Agreement

SDLC Area	Selected Tool	License / Cost	Custom Build Needed?
Planning / PM	Plane (self-hosted)	AGPL-3.0, free (Community Edition)	No — deploy via Docker; configure BYOK AI key
Architecture / Wireframing	Mermaid (diagrams) + Penpot (UI)	Open-source, free	No, but may need a shared component library set up in Penpot
Coding agent	Cline (VS Code + CLI)	Apache-2.0, free (BYOK model cost only)	No — install extension/CLI, connect API keys
Documentation	Markdown docs-as-code in Git + Mintlify or GitBook front-end	Free tier available	Minor — initial doc structure/templates
Testing / QA	Playwright + AI-assisted script generation	Open-source, free	Yes — initial test scaffolding for the pilot project
Maintenance / Observability	Langfuse (self-hosted)	MIT core, free self-hosted	No — Docker deployment
RAG Knowledgebase	Chroma/Qdrant + LlamaIndex + BYOK embeddings	Open-source, free (pay only for embedding/generation API calls)	Yes — this is the project's own custom build

2.2 Installation & Deployment Sequence

- **Day 1–2:** Stand up Plane (Docker) for project tracking; migrate the Week 1 research backlog into it so the rest of the project is tracked inside the tool being evaluated.

11th Floor CyberOne., Eastwood City Cyberpark, Bagumbayan, Quezon City, Philippines 1110
Tel. (632) 7217 0372 Fax. (632) 89126745 E-mail: info@eacomm.com Web: www.eacomm.com

- **Day 2–3:** Install Cline in each team member's VS Code, connect BYOK keys for the chosen models (see section 1.2), and agree on Plan/Act approval conventions so AI-written code is consistently reviewed before merge.
- **Day 3–4:** Stand up Chroma or Qdrant locally/in a container; set up the LlamaIndex ingestion pipeline pointed at the Week 1 research documents as the first knowledgebase content.
- **Day 4–5:** Deploy Langfuse (Docker Compose) and wire it into the RAG pipeline so every query/retrieval is traced from day one, not bolted on later.
- **Day 5:** Set up the documentation repo structure (Markdown + chosen front-end) and the Playwright test scaffold, ready for the Week 3 pilot project.

2.3 Integration Points

The RAG knowledgebase is the connective layer across the stack rather than a stand-alone tool:

- Cline can query the knowledgebase via an MCP server wrapping the RAG API, so the coding agent can pull project-specific context (prior decisions, coding conventions) before writing code.
- Plane's native MCP server can expose project/issue data as another retrievable source, so the knowledgebase can answer "what did we decide about X" using the actual PM record, not just static docs.
- Langfuse traces both the coding agent's tool calls and the RAG system's retrieval steps, giving one observability surface for the whole stack rather than two disconnected ones.

2.4 As-Built Stack — What the Team Actually Deployed

Section 2.1 records the stack proposed on the strength of the Week 1 research. At the team agreement meeting some of those choices were substituted for tools that were faster to stand up or fully open-source, and the RAG generation layer was moved to a free local model. The proposal is kept above as written; the table below records what was actually installed and demonstrated, so the two can be read side by side.

SDLC Area	Proposed in 2.1	Actually Deployed in Week 2	Owner
Planning / PM	Plane (self-hosted via Docker)	Plane, free hosted tier (workspace + project board)	Harlyn Nebreja
Architecture / Wireframing	Mermaid (diagrams) + Penpot (UI)	draw.io for the architecture diagram + Figma for wireframes; Penpot retained as the open-source alternative	Jenilyn Zaulda

11th Floor CyberOne., Eastwood City Cyberpark, Bagumbayan, Quezon City, Philippines 1110
Tel. (632) 7217 0372 Fax. (632) 89126745 E-mail: info@eacomm.com Web: www.eacomm.com

SDLC Area	Proposed in 2.1	Actually Deployed in Week 2	Owner
Coding agent	Cline (VS Code + CLI), BYOK	Cline in VS Code with a BYOK key — unchanged from the proposal	Daryll Tupaz
Documentation	Markdown docs-as-code + Mintlify / GitBook front-end	MkDocs Material, run locally (pip install, mkdocs serve) — no hosted service account needed	Harlyn Nebreja
Testing / QA	Playwright + AI-assisted script generation	Playwright via pytest-playwright; Keploy identified for API regression tests	Vincent Acha
Maintenance / Observability	Langfuse (self-hosted)	SigNoz (Docker), with Playwright's built-in HTML report as the lightweight fallback dashboard	Vincent Acha
RAG Knowledgebase	Chroma / Qdrant + LlamaIndex + BYOK embeddings	Cognexa — the team's own custom platform: ChromaDB semantic retrieval with a local Ollama model or a BYOK external provider (see Section 4)	Reymel Mislang

2.5 Completed Deliverables by Owner

All five Week 2 deliverables were completed and demonstrated. Each owner installed the tool for their SDLC area, built a small working piece, and logged their setup steps in the shared documentation structure.

2.5.1 Development & Implementation — Daryll Tupaz

Cline was installed as a VS Code extension and configured in BYOK mode, so the agent is model-agnostic and the team pays only for what the model provider charges — a local model via Ollama or LM Studio can be substituted at no cost.

- **Installed:** Cline extension in VS Code, API provider configured, model selected for multi-file work with a cheaper model available for routine tasks.
- **Built:** a real coding task run end to end — the agent read the project files, wrote the code, ran it in the terminal, and iterated until it worked.
- **Demonstrated:** the full read–edit–run loop with the approval gate before each action, which is the control point that keeps a human accountable for anything the agent writes.
- **Backup path recorded:** Kilo Code (zero-markup BYOK gateway) or OpenCode in the terminal; Aider as the fallback for a git-centric workflow.

2.5.2 Architecture, Design & Wireframing — Jenilyn Zaulda

Two artefacts were produced so the rest of the team had something concrete to build against: a workflow architecture diagram and a set of wireframes for

11th Floor CyberOne., Eastwood City Cyberpark, Bagumbayan, Quezon City, Philippines 1110
Tel. (632) 7217 0372 Fax. (632) 89126745 E-mail: info@eacomm.com Web: www.eacomm.com

the screens the project needs.

- **Installed:** draw.io in the browser (no local install) and a free Figma design file; Penpot noted as the fully open-source substitute if the team needs to self-host.
- **Built:** an architecture diagram showing how Cline, the RAG knowledgebase, the project board and the QA layer connect, plus wireframes for the knowledgebase question-and-answer screen and the project dashboard.
- **Demonstrated:** a walkthrough of the diagram and each wireframe, explaining what the screens do and where each team member's piece attaches.

2.5.3 Planning, PM & Documentation — Harlyn Nebreja

This area produced the two shared surfaces the rest of the project runs on: a project board that makes tasks and owners visible, and a single documentation structure where every member records their setup steps in the same place.

- **Installed:** a Plane workspace and project on the free hosted tier; MkDocs Material installed locally and served from localhost for the docs site.
- **Built:** a board carrying one task per member's Week 2 deliverable with owners assigned and a Week 2 milestone, plus a documentation tree with a Setup Log page per SDLC area.
- **Demonstrated:** the live board with tasks, owners and milestone, and the docs structure with a page per area.
- **Noted for Week 3:** GitHub Projects remains the lighter alternative, and it sits next to the repository the custom code will be pushed to.

2.5.4 Testing, QA & Observability — Vincent Acha

Two pieces were stood up: a test harness that proves the code behaves, and one tracked signal that shows the system's health rather than just its correctness.

- **Installed:** Playwright through pytest-playwright with browsers provisioned; SigNoz brought up under Docker for traces, metrics and logs.
- **Built:** a small test asserting that the RAG application returns an answer, run in the harness with a visible pass result, and one tracked signal — test pass rate and request latency.
- **Demonstrated:** the harness running live and the tracked metric on a dashboard.
- **Fallback exercised:** SigNoz under Docker is heavy for a laptop, so Playwright's built-in HTML report was kept ready as the demo dashboard —

11th Floor CyberOne., Eastwood City Cyberpark, Bagumbayan, Quezon City, Philippines 1110
Tel. (632) 7217 0372 Fax. (632) 89126745 E-mail: info@eacomm.com Web: www.eacomm.com

it already reports pass/fail and timing per test. Keploy was identified for auto-generating API regression cases.

2.5.5 RAG Knowledgebase — Reymel Mislang (main custom build)

This was the project's only substantial custom build. The system retrieves the most relevant passages from the team's own project documents and then has a local model generate an answer using only those passages, so answers are grounded in the team's research rather than in the model's general knowledge.

- **Stack deployed:** LlamaIndex for orchestration, ChromaDB as the vector store, all-MiniLM for embeddings, and a small model pulled through Ollama for generation — the whole pipeline runs locally at no cost, and no project data leaves the machine.
- **Built:** an ingestion step that chunks and embeds the project documents into the vector store, and a query path that retrieves the top matching passages and returns an answer with its source.
- **Demonstrated:** several real questions answered from the project research with the source passage shown, plus one deliberately off-topic question that the system declined to answer — the negative case is what proves the answers are coming from the knowledgebase and not from the model's training data.
- **Integration noted:** how the knowledgebase is exposed to Cline so the coding agent can look up the team's own project context before writing code.
- **Full detail:** the build is named Cognexa and is documented in full in Section 4; its own documentation set and source live in the project repository.

2.6 How the Pieces Connected in Practice

The five deliverables were built as one chain rather than five separate demos, which is how they were presented:

- The project board carried the work and owners for the whole week.
- The architecture diagram and wireframes defined what was to be built.
- Cline built it, pulling project-specific context from the knowledgebase rather than guessing from generic public code.
- The RAG knowledgebase fed Cline the team's own research as grounded context.
- Playwright tested the result and the observability layer watched its health.
- Every step was logged back into the shared documentation structure.

Stated as one line: the team deployed an open-source, bring-your-own-key

11th Floor CyberOne., Eastwood City Cyberpark, Bagumbayan, Quezon City, Philippines 1110
Tel. (632) 7217 0372 Fax. (632) 89126745 E-mail: info@eacomm.com Web: www.eacomm.com

toolchain across the whole SDLC — planned, designed, built, grounded by its own knowledgebase, and tested — with no vendor lock-in at any layer.

2.7 Deviations From the Proposed Stack, and Why

Four substitutions were made against Section 2.1. Each was a deliberate trade of research preference for something faster to stand up or cheaper to run, and none of them breaks the project's two rules of open-source and model-agnostic tooling.

- **draw.io + Figma instead of Mermaid + Penpot.** Both run in the browser with nothing to install, which mattered for producing wireframes quickly; Mermaid remains the better fit for diagrams that need to be versioned in Git, and Penpot is still the open-source route if the design layer needs self-hosting later.
- **MkDocs Material instead of Mintlify or GitBook.** MkDocs is fully open-source and runs from a local pip install with no hosted-service account, which fits the project rules more cleanly than a commercial documentation front-end with a free tier.
- **SigNoz instead of Langfuse.** SigNoz covers general application observability — traces, metrics and logs — rather than LLM-specific tracing. It is the broader fit for watching the deployed stack, at the cost of being heavier to run locally; Langfuse remains the better choice if the priority shifts to inspecting prompt-level RAG behaviour.
- **Plane's free hosted tier instead of a Docker self-host.** This saved setup time in a one-week window. Self-hosting via docker compose (roughly 2 CPU and 4 GB RAM) is still the open-source path and can be adopted without changing how the board is used.
- **A free local model for RAG generation instead of BYOK cloud embeddings.** Running all-MiniLM embeddings and a small Ollama model locally cost nothing and kept project documents on the machine. It also demonstrates the model-agnostic principle in the strongest form: the generation layer can be swapped for a frontier model through the same interface when answer quality matters more than cost.

Week 2 status

All five deliverables were installed, built, demonstrated, and logged in the shared documentation structure. The deployed stack — Plane, draw.io and Figma, Cline, MkDocs, Playwright and SigNoz, and the LlamaIndex/Chroma knowledgebase — is the stack that carries into Week 3 for the end-to-end pilot project, the manuals, and the GitHub upload of the custom RAG code.

Week 3: Document and Test

Week 3's scope is: document and create manuals for the tools built/deployed,

11th Floor CyberOne., Eastwood City Cyberpark, Bagumbayan, Quezon City, Philippines 1110
Tel. (632) 7217 0372 Fax. (632) 89126745 E-mail: info@eacomm.com Web: www.eacomm.com

upload custom code to GitHub, build a simple end-to-end pilot project using the deployed stack, assess how easy/hard the process was, and present findings.

3.1 Documentation Plan

- Admin/setup manual: step-by-step deployment instructions for each tool in the stack (Plane, Cline, Chroma/Qdrant + LlamaIndex, Langfuse, Playwright), written first as an LLM-generated draft from the actual Docker/config files used in Week 2, then edited by the team for accuracy.
- User manual: how a new team member queries the knowledgebase, submits a coding task to Cline, and reads a Langfuse trace — aimed at someone joining the project who wasn't present for Weeks 1–2.
- Custom source code: any glue code (the MCP wrapper around the RAG API, ingestion scripts, CI config) documented with inline comments plus a top-level README, and pushed to GitHub as the scope of work requires.

3.2 End-to-End Pilot Project

To generate a genuine assessment (not just documentation of the tools in isolation), the team should run one small, real feature through the entire stack: planned in Plane → architecture/wireframe sketched with Mermaid/Penpot → built with Cline → tested with the Playwright/AI test-generation pairing → documented via the docs-as-code pipeline → monitored in Langfuse → with the RAG knowledgebase queried at each stage for prior context. A small internal tool (for example, a CLI or web page that queries the RAG knowledgebase itself) is a good candidate pilot, since it exercises the whole stack while producing something genuinely useful to keep.

3.3 Assessment Framework

For the "feedback and assessment on how easy/hard it was" deliverable, score each tool consistently so the findings are comparable:

Dimension	Questions to Answer
Setup friction	How long from zero to a working install? Any undocumented steps?
Model flexibility	Did BYOK work as advertised across providers? Any lock-in encountered?
Output quality	Did AI-generated code/docs/tests need heavy editing, or were they close to final?
Integration cost	How much custom glue code was needed to connect this tool to the rest of the stack?
Cost at this scale	Actual API spend for the pilot project, extrapolated to a full team/month.
Would we keep using it?	A simple yes/no/with-changes verdict per tool, to make the final presentation decisive rather than descriptive.

11th Floor CyberOne., Eastwood City Cyberpark, Bagumbayan, Quezon City, Philippines 1110
Tel. (632) 7217 0372 Fax. (632) 89126745 E-mail: info@eacomm.com Web: www.eacomm.com

3.4 Presentation Outline

- Findings by SDLC phase (mirrors section 1.1 of this report, updated with hands-on results instead of research-only findings)
- Model evaluation results (updated version of the section 1.2 table with the team's own scored observations)
- RAG knowledgebase demo — live query showing retrieval + citation back to source documents
- Pilot project walkthrough — the feature, and where each tool helped or got in the way
- Recommendation: which tools to keep, which to drop, and what a production version of this stack would need beyond what a 3-week pilot could cover

Cognexa — The Project's Custom RAG and SDLC Platform

Every other tool in the stack was installed and configured. Cognexa is the one system the team wrote, and it is the component the rest of the toolchain leans on. It is a hybrid platform: it runs either fully self-hosted with a local LLM through Ollama and a local database, or deployed to the cloud with a bring-your-own-key external AI provider, and every feature works the same either way. That property is the project's two ground rules — open-source-first and model-agnostic — expressed as architecture rather than as policy.

Cognexa combines two products in one application. The first is a Retrieval-Augmented Generation knowledge base: users upload documents or connect external sources, the system indexes them for semantic search, and a chat interface answers questions from the retrieved passages. The second is AI-driven SDLC project management: a user describes a software project in plain language and the platform generates the stages, artifacts, and documentation for it. The source is MIT-licensed and published at github.com/codewithryry/Cognexa, with a maintained documentation set of six guides.

4.1 What It Does

- **Document ingestion.** PDF, DOCX, JPG, and PNG uploads have their text extracted — pypdf for PDF, python-docx for DOCX, and pytesseract OCR for images — then chunked, embedded, and indexed.
- **Retrieval-augmented chat.** A question is embedded and matched against the user's indexed chunks; the retrieved passages are sent to the model as context and the answer streams back token by token, with every

11th Floor CyberOne., Eastwood City Cyberpark, Bagumbayan, Quezon City, Philippines 1110
Tel. (632) 7217 0372 Fax. (632) 89126745 E-mail: info@eacomm.com Web: www.eacomm.com

answer listing the documents it drew from. Search can be scoped to specific documents, and conversations are saved as named, resumable sessions.

- **Report generation.** A saved chat session or a free-form topic query can be turned into a structured written report, exportable as .txt, .md, .docx, or .pdf, and cached for instant reload.
- **AI-powered SDLC projects.** A plain-English project description is turned into a full lifecycle pipeline with concrete artifacts, generated in the background and tracked on an interactive dashboard with downloadable deliverables.
- **External sources and channels.** Google Drive folders and GitHub can be connected as content sources, and a Telegram bot can be connected so the same knowledge base is reachable outside the web app.
- **Per-user isolation.** Each account sees only its own documents, chat history, and settings, and each user's vectors live in their own isolated collection.

4.2 From the Week 2 Prototype to the Platform

The Week 2 deliverable was deliberately small: a Python web application over the team's own research documents, with the ingestion and query entry points named to match the Week 2 guide (ingest.py and query.py) and lightweight keyword-weighted retrieval instead of embeddings. It already carried the behaviours that matter — answers cited their source passage, low-scoring questions were declined rather than guessed at, unhelpful answers could be flagged, and a pytest suite of golden questions ran as a regression check, which is what caught a retrieval defect during the build.

That prototype proved the idea in days. The platform that followed kept the same contract — retrieve first, cite the source, answer only from context — and replaced the mechanism: keyword scoring became sentence-transformer embeddings in a vector store, the single script became a two-tier application with accounts and a web front end, and question answering was joined by report generation and the SDLC module. The progression is itself a finding: a small retrieval system is enough to validate the approach inside a week, and the semantic and multi-user machinery can be layered on afterwards without changing what the system is for.

4.3 Architecture and Technology Stack

Cognexa is a two-tier web application. A Next.js single-page front end talks to a FastAPI backend over JSON and Server-Sent Events; the backend keeps structured data in MySQL and embeddings in a local, on-disk ChromaDB, and calls either a local Ollama runtime or a connected external provider for

11th Floor CyberOne., Eastwood City Cyberpark, Bagumbayan, Quezon City, Philippines 1110
Tel. (632) 7217 0372 Fax. (632) 89126745 E-mail: info@eacomm.com Web: www.eacomm.com

generation.

Layer	Technology
Frontend	Next.js 16 (App Router), React 19, TypeScript 5, Tailwind CSS v4
Backend	FastAPI (Python) on the Uvicorn ASGI server
ORM / driver	SQLAlchemy 2.0 with PyMySQL
Relational database	MySQL — users, documents, settings, integrations, sessions, projects
Vector database	ChromaDB, local and persistent on disk
Embeddings	sentence-transformers (all-MiniLM-L6-v2)
Local inference	Ollama (default model llama3.2)
External inference (BYOK)	OpenAI, Anthropic Claude, Cohere, Google Gemini, OpenRouter, Cline, or a local LM Studio endpoint
Document parsing	pypdf, python-docx, pytesseract with Pillow for image OCR
Auth	JWT bearer tokens (python-jose) with bcrypt password hashing

4.4 Backend Modules and Data Model

The backend is organised by responsibility, which is what makes it documentable and testable rather than one long script:

Module	Responsibility
main.py	FastAPI application and all HTTP/SSE routes; plan, credit, and priority enforcement
models.py	SQLAlchemy ORM models
schemas.py	Pydantic request and response schemas
auth.py	Password hashing, JWT issuing and verification, current-user dependency
crypto.py	Encryption of stored credentials and tokens
database.py	SQLAlchemy engine and session configuration
rag.py	Document chunking, embedding, and ChromaDB indexing and deletion
query.py	Similarity search, streamed generation (local or external provider), and report generation
scheduler.py	Priority background worker pool for indexing and priority slot gate for generation
sdlc.py	API for SDLC projects, stages, artifacts, pipelines, and tool integrations
sdlc_generator.py	Orchestration of the project generation pipeline
sdlc_ai.py	Prompt templates for SDLC generation

11th Floor CyberOne., Eastwood City Cyberpark, Bagumbayan, Quezon City, Philippines 1110
Tel. (632) 7217 0372 Fax. (632) 89126745 E-mail: info@eacomm.com Web: www.eacomm.com

Module	Responsibility
google_drive.py	OAuth flow, folder selection, and background Drive sync
telegram.py	Bot connection, webhook handling, and session sync
services/	Transactional email delivery via Gmail API or Resend

The relational schema is defined by the ORM models: users and their per-user settings; documents with extracted preview and chunk count; saved AI-provider integrations, data-source connections, and chat channels; chat sessions and the message turns inside them; cached generated reports; and the SDLC projects, stages, and artifacts with the generation status used for progress tracking. Schema changes are applied as ordered, numbered migration files, so an existing installation can be brought up to a newer build rather than rebuilt.

4.5 How the Core Flows Work

- **Upload and indexing.** The backend first checks plan limits and rejects byte-identical re-uploads using a content hash. Text extraction runs synchronously so the response can carry a preview; the heavy chunk-embed-store step is handed to the priority worker pool so the request returns immediately, and the document shows as Processing until the background worker finishes and the chunk count is written.
- **Chat and retrieval.** The question is embedded and matched against that user's collection only. Retrieval deliberately pulls a wider candidate pool and caps how many chunks any one document may contribute, so a question spanning several documents gets context from all of them instead of only the closest match. Metadata questions such as "list my documents" are answered from the database directly, bypassing vector search. Generation streams back as Server-Sent Events — sources first, then answer tokens — and the full exchange is saved to the session.
- **Report generation.** The backend gathers context either from a saved session's history or from a fresh similarity search over the chosen documents, asks the configured provider to synthesise a structured report, then persists it with its source list and returns it for export.

4.6 The AI-Powered SDLC Project Module

This is the second half of the platform and the part that reaches furthest beyond a conventional knowledge base. Rather than filling in a rigid project form, the user writes a description of what they want to build, optionally attaches requirement documents, and optionally picks a provider and model; the platform interprets that into a project definition and generates the lifecycle around it.

11th Floor CyberOne., Eastwood City Cyberpark, Bagumbayan, Quezon City, Philippines 1110
Tel. (632) 7217 0372 Fax. (632) 89126745 E-mail: info@eacomm.com Web: www.eacomm.com

- **Background generation with live progress.** Generation runs as a background job, with a five-step tracker — analysing requirements, generating stages, creating artifacts, generating documentation, finalising — and a progress bar on the project list, which polls while any project is still generating.
- **Pipeline navigator.** A completed project presents a six-node pipeline — Plan, Design, Build, Test, Document, Deploy — where selecting a stage shows its generated artifacts. Stage status can be moved between pending, in progress, review, completed, and blocked.
- **Artifact handling.** Each artifact can be viewed, edited inline and saved, regenerated, deleted, or downloaded as .docx — or as a .zip for generated test suites — and the whole project can be downloaded at once.
- **Tools workspace and Tools Hub.** A per-project workspace offers phase tabs with quick-links into the knowledge base and a checklist, alongside a project-independent directory of recommended tools grouped by SDLC phase and filterable by built-in versus external.
- **API surface.** The module exposes its own routes for projects, stages, artifacts, pipelines and their executions, bring-your-own-key tool integrations, and an activity feed.

4.7 Integrations and Access Channels

- **AI providers (BYOK).** A guided setup dialog walks a new user through choosing local Ollama or connecting their own key for an external provider, and a specific saved integration can be selected per question. Adding a provider is a configuration change in one place rather than a rewrite.
- **Google Drive.** Connected through OAuth 2.0 authorisation code flow, with folders chosen via the Google Picker and synced in the background with live progress. Access tokens are encrypted at rest and refreshed automatically, and more than one Drive account can be connected per user.
- **Telegram.** A bot created through BotFather is connected so the same knowledge base and the same RAG pipeline can be reached as a chat outside the web app; messages arrive by webhook and the conversations sync back into the web app’s chat sessions. Bot tokens are stored encrypted.
- **GitHub.** Connected as a data source so repository content can be indexed alongside uploaded documents.
- **Email.** Welcome, first-login security, and password-changed notifications are sent in the background through the Gmail API or Resend, with automatic fallback between the two and independently toggleable categories.

11th Floor CyberOne., Eastwood City Cyberpark, Bagumbayan, Quezon City, Philippines 1110
Tel. (632) 7217 0372 Fax. (632) 89126745 E-mail: info@eacomm.com Web: www.eacomm.com

4.8 Plans, Credits, and Priority Scheduling

Because the platform is multi-user and a single local model is a shared resource, usage is bounded by plan and contention is resolved by priority rather than by first-come-first-served:

Plan	Price	Documents	Storage	AI usage	Providers
Community	Free (self-hosted)	Up to 25	15 MB	Unlimited on the local model; 50 questions/month via free OpenRouter models	1
Pro	$19/month	Up to 100	10 GB	Unlimited, billed through the user's own provider	Up to 3
Unlimited	$49/month	Unlimited (fair use)	Unlimited (fair use)	Unlimited, billed through the user's own provider	Unlimited

Both document indexing and chat generation are served in priority order proportional to tier, so paid plans are not starved under concurrent load while free-tier users are still served. Plan limits also gate provider, data-source, and chat-channel connections, and monthly AI credit usage is tracked per account.

4.9 Security and Multi-Tenancy

- Passwords are hashed with bcrypt, and every route except health and the register and login endpoints requires a valid JWT bearer token.
- Every resource is scoped by user identity, and each user's vectors are held in an isolated collection, so no endpoint can reach another account's data.
- Third-party credentials — Drive tokens, bot tokens, provider keys — are encrypted at rest.
- The front end logs out automatically after an hour of inactivity.
- In the self-hosted configuration, documents, embeddings, and chat history stay on the deployment; nothing leaves it unless the user deliberately connects an external provider. This is the property that makes the system usable for proprietary source code and internal documents, which is precisely the governance concern the project set out to address.

4.10 Running the System

- **Prerequisites:** Python 3.11 or later with a virtual environment, Node.js for the front end, a local MySQL database, and Ollama with a model pulled. Tesseract is optional and only needed for OCR on image uploads.
- **Database:** apply the base schema first, then the numbered migrations in order.
- **Backend:** activate the virtual environment and run the application with uvicorn on port 8000; interactive API documentation is served at /docs.

11th Floor CyberOne., Eastwood City Cyberpark, Bagumbayan, Quezon City, Philippines 1110
Tel. (632) 7217 0372 Fax. (632) 89126745 E-mail: info@eacomm.com Web: www.eacomm.com

- **Frontend:** install the Node dependencies and start the development server on port 3000, pointing it at the backend through an environment variable if it runs elsewhere.
- **Switching models:** the LLM and embedding model are per-user settings read at request time, so changing model — or moving from a local model to an external provider — needs no code change at all.

4.11 Testing, Documentation, and Licensing

Testing is documented across four levels: unit tests for backend logic such as the priority worker ordering and plan-permission checks, integration tests against a real MySQL and ChromaDB instance, end-to-end tests driven through the browser, and manual exploratory passes for layout, theme, and dialog behaviour. The test catalogue holds well over a hundred numbered cases across fourteen functional areas — authentication, upload and limits, knowledge base, chat and retrieval, chat integrations, reports, data sources and channels, Drive OAuth, Telegram, AI credits, billing, data management, priority scheduling, vector isolation, email, and the SDLC module — with priorities and explicit entry and exit criteria gating a release.

The documentation set in the repository is kept in sync with each release and covers the product from six angles: software documentation, user manual, administrator manual, testing documentation, technical documentation, and the SDLC platform in depth, alongside a written case study and a tools assessment. The contribution guide states the same two ground rules the project began with — open-source-first and model-agnostic BYOK — and requires that any behaviour change update the matching document in the same pull request. The project is released under the MIT licence.

4.12 Where Cognexa Sits in the Team Toolchain

Cognexa is not a sixth parallel demo; it is the layer the other four connect through, and the place where the project's findings ended up as working software:

- It supplies the coding agent with project-specific context — Cline is one of the providers it can be connected to — so generated code follows the team's own decisions rather than generic public patterns.
- It is the application the QA layer tests against and the source of the metrics the observability layer watches.
- Its own Tools Hub recommends, by SDLC phase, the same class of tools the team deployed in Week 2 — design, coding, testing, and observability — which makes the platform a working index of the project's research rather than a separate artefact.

11th Floor CyberOne., Eastwood City Cyberpark, Bagumbayan, Quezon City, Philippines 1110
Tel. (632) 7217 0372 Fax. (632) 89126745 E-mail: info@eacomm.com Web: www.eacomm.com

- Its manuals and setup logs feed the shared documentation structure, and its source is the custom code published to GitHub for Week 3.
- Its core design — retrieve first, cite the source, answer only from context, and keep the model swappable — is the working answer to the Week 1 research question about how to ground an AI workflow in an organisation's own knowledge without surrendering control of the data.

Summary of Recommendations

Area	Recommendation
Coding agent	Cline (open-source, Apache-2.0, BYOK, most auditable approval workflow)
Models	Claude for architecture/reasoning/documentation quality; a cheaper high-throughput model for high-volume/boilerplate work; re-evaluate quarterly given how fast benchmarks are moving
Project management	Plane, self-hosted, for native AI + MCP support
Design/wireframing	Mermaid for architecture diagrams (text-based, versionable); Penpot or a BYOK AI wireframe generator for UI screens
Documentation	Docs-as-code (Markdown in Git) with LLM-drafted first passes and human editing
Testing/QA	Playwright with AI-assisted test authoring, not full autonomous AI-native QA at this project size
Observability	Langfuse, self-hosted, covering both the coding agent and the RAG pipeline
Knowledgebase	Chroma/Qdrant + LlamaIndex + BYOK embeddings + hybrid search, with Langfuse tracing from day one

This report is based on research conducted in July 2026. The AI tooling landscape in this space is changing on a roughly monthly cadence — several tools referenced here changed status (shut down, forked, or repositioned) during the research window itself — so tool selections in section 2.1 should be re-verified against current repository/vendor status immediately before Week 2 installation begins.

11th Floor CyberOne., Eastwood City Cyberpark, Bagumbayan, Quezon City, Philippines 1110
Tel. (632) 7217 0372 Fax. (632) 89126745 E-mail: info@eacomm.com Web: www.eacomm.com