ZUSTIS • RESEARCH

APPLIED RESEARCH / AUTONOMOUS CODING AGENTS

Fourteen Hours, Two Agents, One Deployed Tool.

How a heterogeneous planner–executor–reviewer loop shipped curlit v1.3 to production — and why it is a working demonstration of Zustis's validation-first thesis for enterprise AI.

AUTHOR

Abhimanyu Singhal
Founder, Zustis Technologies Ltd.

PUBLISHED

April 2026
zustis.com / research

ARTIFACTS

curlit.zustis.com
github.com/abhimanyusinghal/curlit

APPLIED RESEARCH — ZUSTIS

Fourteen hours, two agents, one deployed tool

By **Abhimanyu Singhal**, Founder, Zustis Technologies Limited (DIFC) · April 2026

In mid-April 2026, we built and shipped a working developer utility called **curlit** in **fourteen hours of active build effort** — from blank repository to production deployment at curlit.zustis.com, open-sourced at github.com/abhimanyusinghal/curlit, now at version 1.3. The novel element is not the tool itself, though the tool is useful. The novel element is how it was built. **No human hands wrote the implementation code, and no human was a gating step between features.** Two frontier models — **Claude Opus 4.6 as executor and GPT-5.4 as planner-and-reviewer** — worked in a structured planner/executor/reviewer loop, with validation driven primarily by automated end-to-end Playwright tests generated, reviewed, and executed inside the loop itself. Human involvement was deliberately minimised: the single human architect set product intent, arbitrated the rare disagreements between the two agents, conducted exploratory review of finished increments, and approved final version cuts. No line-by-line code review. No manual regression sweep. No step-by-step babysitting. The output is production-running software that solves a real problem for real users.

We are publishing this case study because **curlit is applied research**, not a product launch. At Zustis, we hold a specific technical thesis about enterprise AI: outputs must be validated against reality, not trusted on authority. Most of the conversation about autonomous coding agents in 2025–2026 has stayed at the level of demos, benchmarks, and vendor marketing. What has been missing is **disciplined, reproducible evidence** of the dual-agent pattern deployed to ship working software, with the architectural reasoning made explicit. This article supplies that evidence and the reasoning behind it.

What curlit is and why we built it

Curlit sits in a crowded but persistently under-served niche: the space between a quick `curl` one-liner and a full API client like Postman or Bruno. Developers live in cURL because it is ubiquitous, scriptable, and close to the wire — but raw cURL is hostile when you want to iterate on requests, compose headers cleanly, share reproducible invocations with teammates, or keep a lightweight log of what worked and what did not. Heavier API clients solve those problems but drag in workspaces, accounts, sync, and UI weight that many engineers actively avoid. **Curlit is the minimum viable bridge**: a fast, shareable way to build, execute, and revisit cURL-style HTTP invocations with just enough structure to be useful and not enough to be in the way.

That is a small product idea. It is deliberately small. We chose it because a small, honest scope was the right substrate for a methodology experiment. The tool had to be genuinely useful — otherwise the exercise is vanity — but it had to be bounded enough that fourteen hours was credible. The repository is open under the Zustis sponsorship, the deployment is live, and the code is inspectable. What follows is how it was produced.

The dual-agent pattern, concretely

Curlit was built using a **planner/executor/reviewer split across two heterogeneous frontier models**. The roles were assigned as follows:

GPT-5.4 acted as planner and reviewer. Before any code was written, GPT-5.4 was given the product intent and asked to decompose it: interface contracts, module boundaries, data flow, edge cases, failure modes, and an ordered task list with acceptance criteria. After each implementation increment, GPT-5.4 was given the diff, the task description, and the running product state, and was asked to review — not to rubber-stamp, but to find what was wrong, what was missing, what would break under load, and what violated the plan. It flagged regressions, named specific functions to refactor, and occasionally rejected entire increments.

Claude Opus 4.6 acted as the coding executor. Given a task from the planner and the current repository state, it produced the code: file edits, new modules, test scaffolding, configuration, commit messages. It ran its own sanity checks, proposed its own tests, and exposed its reasoning so the reviewer had something substantive to critique.

The human architect — held four light-touch responsibilities: setting product intent at the start, arbitrating the rare disagreements between the two agents, conducting exploratory spot-checks of finished increments, and approving final version cuts. Everything else ran inside the loop. Validation between increments was automated: Playwright-driven end-to-end tests were generated by the executor as part of each feature, reviewed by the planner for coverage and edge cases, and executed against every landed change. A feature that did not pass its E2E suite did not count as landed. **The human was not a gating step at each commit — the automated test layer was.** The distinction matters for the methodology claim. What we are describing is an autonomous loop with exploratory human oversight, not a human-in-the-loop pipeline with AI assistance. The former is the genuine research contribution. The latter is widely deployed and well understood.

This split maps onto an emerging consensus in the 2025–2026 agent literature. The Plan-then-Execute pattern, formalized in Del Rosario, Krawiecka, and Schroeder de Witt's September 2025 paper on resilient agent architectures, argues that **separating strategic planning from tactical execution yields systems that are easier to audit, more resilient to control-flow hijacking, and measurably more cost-efficient** than reactive patterns like ReAct. Plan-and-Act, from Berkeley and Meta, showed the same separation hitting state-of-the-art on WebArena-Lite by using a dedicated planner LLM. AgentCoder went further and showed that generating tests in a *separate agent* that cannot see the implementation code beats generating them in the same agent that wrote it — 96.3% pass@1 on HumanEval with GPT-4, above every prior multi-agent framework at the time.

What curlit adds to this literature is heterogeneity. Most published multi-agent frameworks use the same base model in every role. We deliberately did not.

Why two different models matter

The common shorthand in 2026 developer discourse — “Claude for coding, GPT for reasoning” — is too coarse to be useful. Looking at the actual benchmark landscape as of April 2026, a more honest picture emerges.

Claude Opus 4.6, released February 5, 2026, scores 80.8% on SWE-bench Verified and 65.4% on Terminal-Bench 2.0 per Anthropic's system card, and it dominates long-context coherence benchmarks — 76% on MRCR v2 at one million tokens, versus 18.5% for GPT-5.2 in the same evaluation. It is especially strong at **long-horizon agentic coding, multi-file reasoning, and intent recovery from ambiguous prompts**. Practitioners, including Nathan Lambert, have noted that switching away from Opus on vague specifications immediately increases babysitting load.

GPT-5.4, released March 5, 2026, is the first mainline OpenAI model to absorb the Codex line's frontier coding capabilities. It scores 57.7% on SWE-bench Pro — the harder, novel-repository evaluation where top models cluster in the 40s and 50s — and 75.0% on OSWorld-Verified, surpassing human expert performance. Its particular strengths are **planning quality on harder problems, tool-use breadth, computer-use reliability, and cost** ($2.50/$15 per MTok against Opus's $5/$25).

The rational mapping is therefore not “Claude does everything” or “GPT does everything.” It is: **give each model the job at which it is measurably better**. GPT-5.4's relative advantage on structured planning, harder-problem reasoning, and breadth makes it the stronger planner-reviewer. Opus 4.6's relative advantage on multi-file coherence, long-context agentic execution, and ambiguous-intent recovery makes it the stronger implementer. The pairing is complementary, not redundant.

This is what the ensemble literature has been quietly converging on. *Beyond Single LLMs* (Chen et al., arXiv: 2510.01379) studied seventeen frontier models across five languages on HumanEval-X and showed that multi-stage performance-guided orchestration reaches 91.5% pass@1 on Rust versus GPT-4o's 69.5% — not because any one model was better, but because different models were better at different stages. *EnsLLM* (arXiv: 2503.15838) showed cross-model voting with CodeBLEU and CrossHair differential behavior reaching 90.2% on HumanEval and 50.2% on LiveCodeBench, above any single contributor. *More Agents Is All You Need* (arXiv: 2402.05120) formalized the underlying scaling law. The 2025 paper *Wisdom and Delusion of LLM Ensembles for Code Generation and Repair* made the theoretical case directly: **the assumption that some single best model is enough is provably wrong on any non-trivial distribution of coding tasks**.

> *The dual-agent pattern curlit uses is a minimal, disciplined instance of the ensemble insight — two models, two roles, one feedback loop, one human arbiter.*

Why a separate reviewer is not cosmetic

The single most load-bearing part of the pattern is that **the reviewer is a different model from the executor**. This is not an aesthetic choice. It is a response to a specific, measurable failure mode of single-model self-critique.

Stechly, Marquez, and Kambhampati's 2023 paper, bluntly titled *GPT-4 Doesn't Know It's Wrong*, demonstrated that iterative self-critique within a single model is unreliable — a model's self-evaluation inherits its own blind spots. The 2025 *Finch-Zk* work quantified the alternative: fine-grained cross-model consistency checks improved hallucination-detection F1 by **6 to 39 percentage points** over single-model baselines, with downstream accuracy improvements on GPQA-diamond up to 80.8 percentage points depending on the judge configuration. *Cross-Re-*

fine, a dual-LLM variant of Self-Refine, showed that heterogeneous generator-critic pairs work where homogeneous self-refinement collapses, especially with smaller models. CaLM (arXiv:2406.05365) showed that even a *smaller* secondary model can validate a larger primary model's grounded generation effectively.

The mechanism is straightforward. When the same model both writes and reviews, systematic errors — confident misreadings of a spec, skipped edge cases, hallucinated API shapes, shallow test coverage — survive review because the reviewing pass shares the generating pass's priors. A different model has different priors. It reads the spec differently, cares about different edge cases, and often notices exactly what the first model was predisposed to miss.

On curlit, this showed up concretely. GPT-5.4 repeatedly caught where Opus 4.6 had quietly over-scoped an increment, where error handling was narrower than the task required, where a naming choice would collide with a later-planned module, and where a test was only testing the happy path. Opus 4.6, conversely, caught places where GPT-5.4's plan had underspecified an interface, particularly when the planner's decomposition assumed properties of the runtime that did not hold. Neither model, working alone, would have produced the version currently running at curlit.zustis.com in fourteen hours. The loop is not a tax. It is the mechanism.

Validation-first AI, made operational

This brings us to the Zustis thesis. Our position on enterprise AI is that **outputs must be validated against reality, not trusted on authority**. We call this approach validation-first and reality-dependent, and we contrast it deliberately with the dominant authority-dependent pattern that retrieves text from a trusted corpus and treats a language model's synthesis of that text as the answer.

Authority-dependent AI — the standard RAG pipeline — grounds outputs in retrieved sources but adds no independent check that the model's synthesis is actually faithful to those sources or correct in the world. If the retriever is right and the prompt is well-formed and the model behaves, the output is usually good. When any of those assumptions fails, the failure is silent. The model still sounds confident. The pipeline still runs. There is no second signal.

Validation-first AI inverts the default. It treats model output as a **hypothesis that must earn its status as an answer** by surviving an independent check — tests that execute, a schema that parses, a second model that disagrees productively, a symbolic trace that terminates. The 2025 hallucination survey (arXiv:2510.06265) taxonomized this as the third layer of grounding, above retrieval and reasoning-structure grounding. The November 2025 *Verification-First Reasoning* paper (arXiv:2511.21734) showed that even a pure prompting-level variant of the same instinct — ask the model to verify a candidate answer before committing — delivers meaningful accuracy gains at effectively zero marginal cost.

The dual-agent pattern used to build curlit is a form of validation-first AI. The reviewer is the validator. Its job is to reject, refine, or accept the executor's output against an independent, model-level reading of the plan. When combined with executed tests and a human arbiter, it produces a chain of validation rather than a chain of authority. Each commit that lands has survived: executor self-checks, cross-model review, automated tests, and human acceptance. Each of those is a verifier. No layer is trusted on authority alone.

The primary validator in the curlit loop was the automated end-to-end test suite. Playwright tests were generated by the executor as part of each feature increment, reviewed by the planner for coverage and adversarial edge cases, and executed against every landed change. The validation chain at each commit therefore ran: executor self-checks, cross-model planner review of both implementation and test coverage, executed Playwright E2E suite, and exploratory human spot-check at the end of the work session. The human layer was the lightest of these, not the heaviest. Its function was feedback and direction-correction on product feel, not behavioural verification — the E2E suite did that job. This is the right shape for the methodology: automated verification carries the load, and the human stays in an oversight role narrow enough to not become a bottleneck.

This is the same architectural intuition we apply to enterprise AI systems at Zustis — neuro-symbolic verification, knowledge-engineering-grade schemas, and explicit verifier stages between generation and use. Curlit is the smallest credible demonstration that the intuition ships working software.

Where this sits in the autonomous coding landscape

The broader market in April 2026 is unambiguously agentic. Anthropic reports that the majority of its own code is now written by Claude Code, whose lead-agent-plus-subagents architecture has become a reference pattern. Cognition's Devin 2.0, now paired with the acquired Windsurf IDE, handles 4–8 hour junior-level engineering tickets at the $20/month tier; Litera reports 40% test-coverage gains and 93% regression-cycle reduction after deploying it. GitHub's Copilot coding agent, as of February 2026, lets you assign the same issue to Claude, Codex, and Copilot models simultaneously and compare the resulting pull requests. Cursor's Composer 2, Google's Jules and Antigravity IDE, OpenHands, Factory Droid, Amazon Q Developer, Atlassian Rovo Dev, and the open-weight MiniMax M2.5 and GLM-5 families have all pushed the capability envelope in parallel. SWE-bench Verified scores now cluster in the 80s for frontier models; Claude Opus 4.7, released four days before this article, reached 87.6%.

Curlit is not positioned against any of these. It is a deliberately small artifact demonstrating a specific *architectural pattern* — heterogeneous planner-executor-reviewer with automated E2E validation and exploratory human oversight — applied end-to-end to ship real, deployed, open software on a tight timeline. Most of the products above are coding environments or agent platforms. Curlit is the output of a coding process. The distinction matters. The industry has reasonably-well-resourced answers to *what tools autonomous coding agents should be*. It has far fewer disciplined answers to *how teams should actually use them to ship*.

What Zustis is pioneering

Zustis Technologies Limited, incorporated under DIFC's Innovation and AI framework in Dubai, operates at the intersection of enterprise AI architecture, neuro-symbolic systems, and knowledge engineering. The work we do for clients is not visible in a public repository. Curlit is. It is meant to be.

We are pioneering the **applied implementation of autonomous coding agents inside a disciplined, validation-first methodology**. The language matters. Theorizing about multi-agent code generation is a crowded field with significant academic output. Building an end-to-end coding agent platform is a crowded field with significant

commercial output. What is far less crowded is the middle — teams that take the best available frontier models, assemble them into a specific validated methodology, ship working software, and document the methodology with enough precision that others can evaluate and adopt it.

We treat open source as a research instrument. Curlit is open because **methodology claims about software have to be demonstrable in software**. A repository that a reader can clone, read, and run is a stronger evidentiary basis than a conference slide or a benchmark screenshot. The choice to sponsor curlit as an open artifact under the Zustis umbrella is a choice to put our methodology on the record in a form that can be checked.

Lessons and implications for enterprise AI

Several lessons from the curlit build generalize beyond the tool.

First, heterogeneity is not optional for high-quality autonomous code. Single-model loops — even with the strongest available frontier model — are measurably weaker than cross-model loops on any non-trivial task distribution. Enterprises building internal AI coding platforms that lock to a single vendor are optimizing for procurement convenience at the cost of output quality. The ensemble literature is now clear enough on this that defaulting to single-model pipelines should require justification, not the reverse.

Second, the role of the human architect becomes narrower and more critical, not broader. Our human time on curlit went almost entirely to four light-touch activities: product intent, occasional disagreement arbitration, exploratory review, and final version acceptance. Everything else — including the bulk of the validation work, which lived in the Playwright E2E suite and the cross-model review pass — was machine work. Properly constructed, the loop is autonomous with exploratory human oversight, not human-driven with AI assistance. The difference sounds rhetorical. It is not. It is the difference between a methodology that scales and one that does not.

Third, validation-first is a generalizable enterprise pattern, not just a coding pattern. The same architectural logic — generation by one system, independent validation by a different system, explicit acceptance criteria, human arbitration only at decision points — applies to enterprise knowledge workflows, contract analysis, regulatory compliance, and any domain where silent model failure is the dominant risk. The RAG-only enterprise stack optimizes for recall and confidence. The validation-first stack optimizes for correctness under adversarial conditions. In regulated markets, including the one we operate in under DIFC, the second is the only defensible architecture.

What comes next: closing the last soft layer

Exploratory human review is the final soft layer in the curlit methodology. It is the one place where the loop still depends on a person. Our next methodology increment, already in design, adds a dedicated **automated validator agent** as a third role alongside executor and planner-reviewer. Its sole responsibility is to exercise each finished increment against the original product intent, the E2E Playwright suite, and an adversarially generated test layer — without a human in the verification path. If the validator passes, the increment lands. If it fails, it returns to the executor with specific reproducible failures. The human architect retains only two responsibilities: product intent at the start, and final version acceptance at release boundaries.

That configuration is the path to a **fully autonomous validation-first build loop**: three agents, three roles, three independent verifiers (cross-model review, automated E2E, adversarial validation), and a human present only at the endpoints. Curlit v1.3 is the last version built with exploratory human review in the middle. Everything after is a test of whether a disciplined three-agent loop can hold the validation-first line without a human in the verification path at all. We expect it will. We will publish the results either way, which is the point of doing research in the open.

Conclusion

Curlit is a fourteen-hour autonomous build. It is also a position statement. The position is that **the autonomous coding agent conversation has matured past demos and leaderboards and is now ready for disciplined methodology work** — and that the methodology that generalizes is heterogeneous, planner-executor-reviewer, validated primarily through executed E2E tests and cross-model review, with exploratory human oversight narrow enough to not become a bottleneck. We did not need to invent new models to demonstrate this. We needed to assemble existing frontier models with enough architectural clarity that the output earns its credibility through verified behaviour rather than trusted authority.

Version 1.3 is live, the repository is public, and the methodology it was built with is the same methodology we bring to enterprise AI work at Zustis. The tool is the smallest, clearest proof point we could publish. The methodology is the larger claim. The three-agent validator-in-the-loop extension is the next claim. We intend to keep shipping all of it, in public.

ABOUT THE AUTHOR & SPONSOR

Abhimanyu Singhal is the founder of **Zustis Technologies Limited**, an AI and applied-research firm incorporated under the DIFC Innovation and AI framework in Dubai. Zustis builds enterprise AI systems at the intersection of neuro-symbolic architectures, knowledge engineering, and validation-first design. Curlit was built and open-sourced under Zustis sponsorship as part of the firm's applied research programme on autonomous coding agents.

curlit.zustis.com github.com/abhimanyusinghal/curlit zustis.com