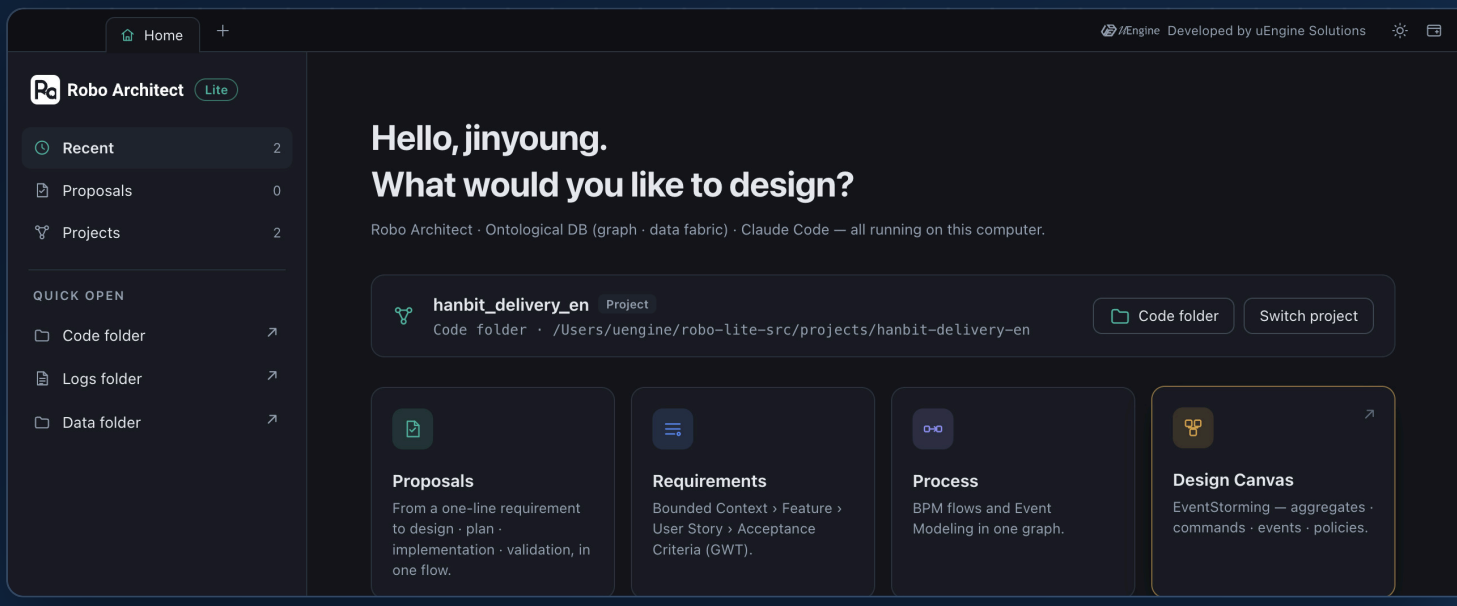
Robo Architect Lite
AI ARCHITECTURE WORKBENCH

PRODUCT OVERVIEW AND ADOPTION PROPOSAL

When requirements change, the design moves with them.

Describe a requirement in plain language. It is refined into a domain-driven design (DDD) with acceptance criteria and planned by your project's design principles. Claude Code on your PC builds it in an isolated workspace, and an accepted change **updates the code and the design graph together**.



21
AI work procedures defined as skill files

7
service containers running on your PC

Dual Merge
code and design graph updated together

October 2026
AI architecture workbench · Desktop app (Lite)

uEngine

CONTENTS

Contents

01 The AI coding paradox 3

02 Specs · scenarios · domains 4

03 Managing specs across teams 5

04 Introducing Robo Architect Lite 6

05 Key deliverables ①② 7-8

06 Product screens at a glance 9

07 Strengths ①② Requirements · Plan 10-11

08 Strengths ③④ Build · Validate 12-13

09 Strengths ⑤⑥ Change impact · Legacy 14-15

10 System architecture and runtime 16

11 Technology stack and skills 17

12 How to use it (real screens) 18-19

13 Security and governance 20

14 Installation and delivery 21

15 Development and quality assurance 22

16 MSAEZ track record · Adoption 23-24

Summary

In AI-assisted development, **defining what to build and how to check it comes first**. Robo Architect Lite links requirements, business scenarios and the domain model in a project graph. Each change is one proposal, carried through design, implementation, validation and merge.

SDD · BDD · DDD

Specs define requirements, scenarios set acceptance criteria, domains set the scope of each design

Proposal-based

Each change is one proposal: requirements, plan, build and validation results in one history

Runs on your PC

The graph DB and analysis services run in Docker; Claude Code on your PC does the AI work

Change management that updates code and design together

When only the code changes, the design documents stop being a reliable reference. Accepting a validated proposal **merges the code and updates the design graph in one step**, so the next proposal is analyzed against the current design.

What this document covers

Product overview

pp. 3–15
Background · deliverables · strengths

System

pp. 16–17
Runtime · tech stack · skills

How to use it

pp. 18–19
Real product screens · 6 steps

Adoption

pp. 20–24
Security · installation · track record · schedule

01 · BACKGROUND

The AI coding paradox

AI writes code fast, but reviewing and fixing that code takes time too. **Clear requirements, business context and acceptance criteria** are where effective use of AI begins.

24% faster
What developers expected
Expected time saved with AI

19% slower
Actual task time
Time to finish tasks with AI

16 · 246
Developers and tasks
Experienced open-source developers

Source: METR, randomized controlled trial (2025). Measured with the tools and working environments of that time.

Four issues to manage when using AI

ISSUE 1

Conveying business context

AI needs the organization's business rules and terms. If they are not organized, people re-explain them in every chat, and gaps lead to wrong code.

ISSUE 2

Review and rework

Generated code still has to be checked against the requirements. Without clear acceptance criteria, review rests on judgment and takes longer.

ISSUE 3

Ambiguous requirements

Experienced developers fill gaps in a spec from domain knowledge. AI without that context can miss the intent, so requirements must be made concrete first.

ISSUE 4

Sharing design rationale

Decisions and reasons left in chat history are hard to find later. Kept with the requirements, they survive handovers and can be reused.

How Robo Architect Lite addresses them

Context
Project graph · design principles · legacy analysis
AI sees the same business context at every stage

Review effort
Acceptance criteria · design structure checks
Each verdict shows its evidence: code and tests

Ambiguity
User stories · Given · When · Then
Made concrete before coding; AI asks when unclear

Rationale
Proposal history · project design principles
Decisions and reasons kept for later changes

Set requirements and acceptance criteria before AI writes code. Managing business rules and design rationale together gives implementation and review a consistent context.

02 · BACKGROUND

Specs, scenarios and domains supply context

Robo Architect connects three methods: **spec-driven development (SDD)** defines requirements, **behavior-driven development (BDD)** describes expected behavior, and **domain-driven design (DDD)** sets the boundaries of business areas and their terms.

SDD · SPEC-DRIVEN

Define the work in a spec

Write the purpose and requirements, then the implementation plan and task list. AI and developers work from the same spec.

BDD · BEHAVIOR-DRIVEN

Describe by example

State condition, action and result: "withdraw $20 from $100, leaving $80." These become acceptance criteria.

DDD · DOMAIN-DRIVEN

Split by business area

Draw domain boundaries from business concepts and rules, with clear terms and duties for each area.

Example: refining a cash withdrawal requirement

Format	Example — ATM withdrawal	What is still missing
User story	"A customer can withdraw cash."	Exceptions such as a low balance, and how to handle them
Requirements spec	"Authenticated users can withdraw cash."	Authentication, withdrawal limits and behavior on failure
Detailed design	`if balance >= amount: balance -= amount`	Hard for non-developers to review
BDD scenario	**Given** $100 · **When** $20 is withdrawn · **Then** $20 is paid out, $80 remains	Business and developers check one example

Where the three methods are used in Robo Architect Lite

SDD
Proposal · Plan · task list
Confirmed before coding

BDD
Given · When · Then criteria
Used to judge results in validation

DDD
Contexts · aggregates · events
Detailed DDD mode has six stages

BDD scenarios let business and IT review rules together. Customers confirm that conditions and results match the intent, and AI writes tests from them. Each scenario is stored in the graph with its design elements and used as a criterion in validation.

03 · BACKGROUND

Specs from many teams need to be linked

File-based specs suit one team's requirements and plans. But **once information spreads across teams and repositories**, aligning terms and tracing the impact of a change get harder. Four issues need managing together.

Issue	Cause	Effect on the work
Scattered specs	Specs live inside each repository	Other teams' work is hidden, so features get built twice
Inconsistent terms	Each repository defines its own terms	Is a “subscriber” the same as a “customer”? Someone must check
Impact analysis	Links exist only as document links	A regulation change means checking documents and code by hand
Knowledge handover	Knowledge sits with people and chat logs	Handovers lose knowledge or repeat analysis

> Robo Architect keeps **how requirements, the domain model, acceptance criteria and change history relate** in a project graph. For a new change, it follows these links to separate new from modified elements and flags possible conflicts with implemented ones.

File-based versus graph-based specs

Aspect	File-based specs	Graph-based specs
Where specs live	Markdown files per repository	One domain graph per project
Boundaries	Repositories and folders	Bounded contexts (business areas)
Terms	Each repository its own	Managed per domain, with relationships
Change impact	File search, asking the owners	Traced along links, with conflict risk
Criteria	Written in documents	Given · When · Then linked to the design
Knowledge	Per-project files	Design elements linked to change history

> **Files and the graph work together.** Code stays in your Git repositories, while Robo Architect manages design relationships and change history as a graph. Implemented code is committed to a branch of the target repository.

04 · OVERVIEW

Introducing Robo Architect Lite

Robo Architect Lite is a **desktop AI architecture workbench for spec-driven development**. It refines requirements into a DDD design with acceptance criteria and plans by your design principles. Claude Code implements in an isolated workspace.

INPUT

Requirements

A sentence or a pasted requirements document. Existing code and DBs can serve as evidence.

PROCESSING

DDD design and plan

Contexts, features, stories and criteria, then aggregates, commands, events and architecture.

OUTPUT

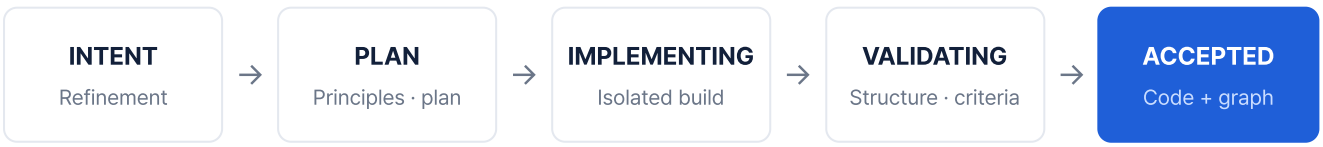
Code and design graph

Code on a dedicated branch with validation results. Accepting updates branch and graph.

How it differs from the usual approach

	Document-centered design	Robo Architect Lite
Design form	Diagrams and documents	Queryable graph (26 node, 32 relationship types)
Unit of change	Separate tickets and documents	One proposal, from requirement to validation
Standards	Documents kept by individuals	Each plan decision cites a design principle
Implementation	Separate from the design	Claude Code, in an isolated branch workspace
Validation	Reviewer's judgment	Criteria and structure checked, with evidence
AI procedures	Personal prompts	Steps and rules in 21 skill files

The flow at a glance



You review each stage before moving on. **The target branch and the design graph change only after final acceptance (Accept).**

05 · DELIVERABLES

Deliverables ① — the project design graph

A proposal's design is stored in the **project graph**. Each project is its own graph inside the graph DB (Ontological DB), so designs stay separate, however many projects you create.

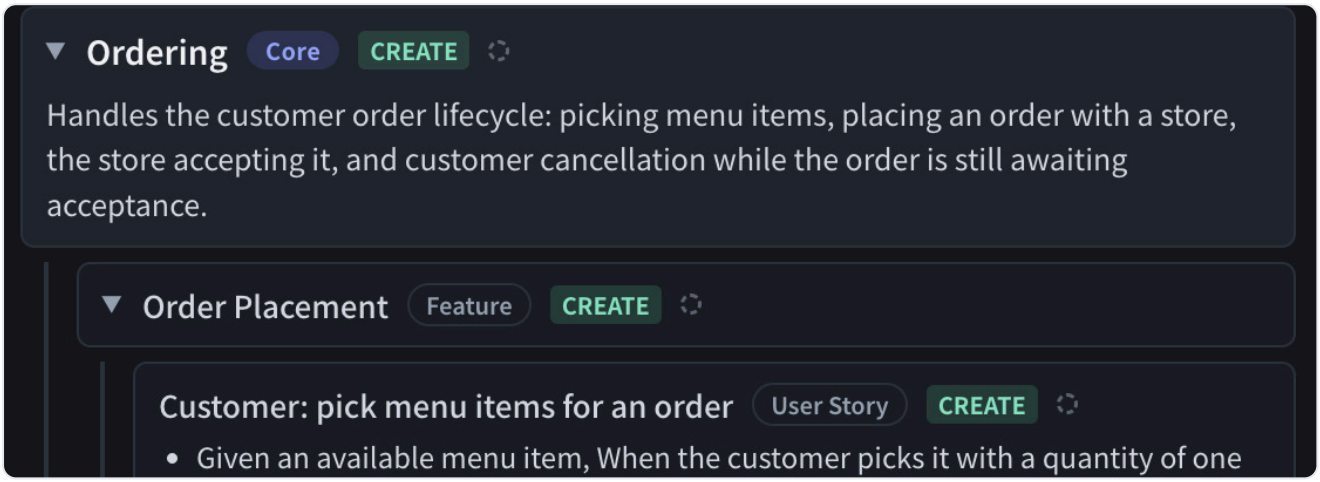


Figure 1. Strategic design — under the bounded context Ordering (Core), the feature Order Placement, a user story and its Given · When · Then acceptance criteria. Each element is marked as new (CREATE). *Demo data (tutorial exercise).*

What the graph stores

Layer	Nodes
Requirements	Requirement · Feature · UserStory · Given/When/Then (acceptance criteria)
Strategic design	BoundedContext (classified as Core · Supporting · Generic) · Policy
Tactical design	Aggregate · Command · Event · ReadModel · Property · Invariant
Screens · flows	UI · Journey · JourneyStep
Changes · standards	Proposal (PRO-NNN) · Constitution (project design principles)

26
Node types
From requirements to proposals and principles

32
Relationship types
EMITS · TRIGGERS · HAS_GIVEN · EFFECT …

1 : 1
Project to graph
A separate graph for each project

Follow the links between design elements to find what you need. A user story's **Design trace** in the Requirements tab shows, in one line, the elements that handle it — for example, Place order button → PlaceOrder (command) → Order (aggregate) → OrderPlaced (event).

05 · DELIVERABLES

Deliverables ② — a record of every stage

Every stage of a proposal leaves a record. To learn later “why does this aggregate exist?”, open the proposal and see **the requirements, the plan's reasoning, the commits and the validation results** together.

Stage	Deliverables	Where it is kept
INTENT	Strategic design (processes · contexts · features · stories · criteria), Impact Map	Proposal
Principles	Markdown: Core Principles · Technology Constraints · Architecture · Repository Strategy · Governance	Project graph (one per project)
PLAN	Tactical design, architecture decisions and their principles, open areas, impact analysis	Proposal
Build	Tasks by phase, one commit per task	`proposal/PRO-NNN` branch
Validation	PASS · FAIL · SKIP cards with their evidence	Proposal
ACCEPTED	Merged code, updated design graph, status history	Target branch · project graph

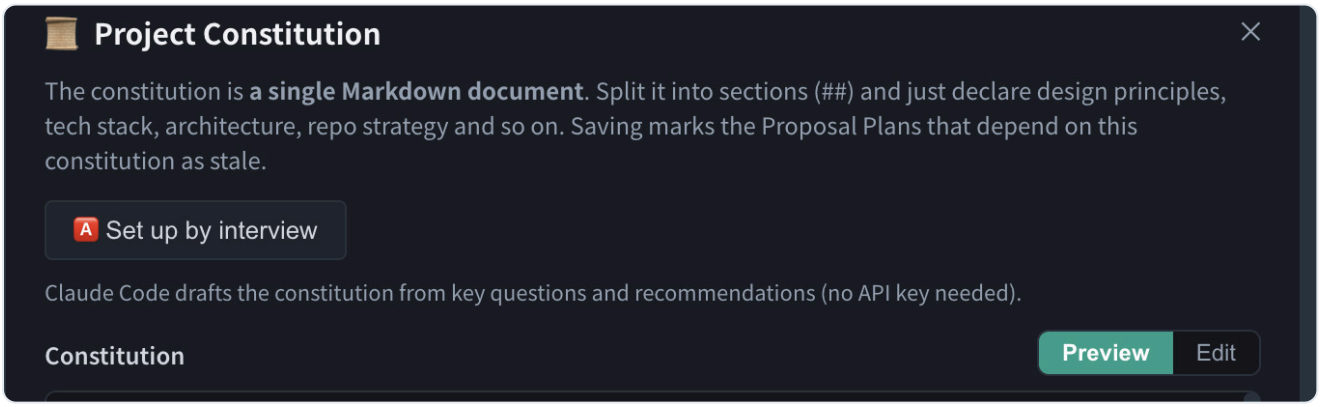


Figure 2. The design principles (Constitution) editor — create the document by interview, then preview or edit it. *Demo data (tutorial exercise).*

Results of tutorial proposal PRO-001 (demo data)

2 → 7
Questions → principles
Two interview questions

18
Tasks · commits
8 phases, a commit per task

125 files
9,290 lines
Excluding the lock file

296
Tests passed
`npm test` after merge

06 · SCREENS

Product screens at a glance

Eight feature cards on Home open each screen as a tab. The Requirements, Process, Design and Data tabs show **one project graph in different views**. Real product screens with demo data (tutorial exercise).

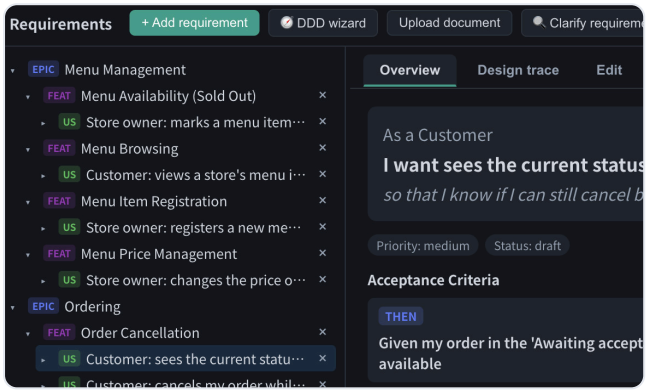


Figure 3. Requirements — context (EPIC) › feature (FEAT) › story (US) and acceptance criteria

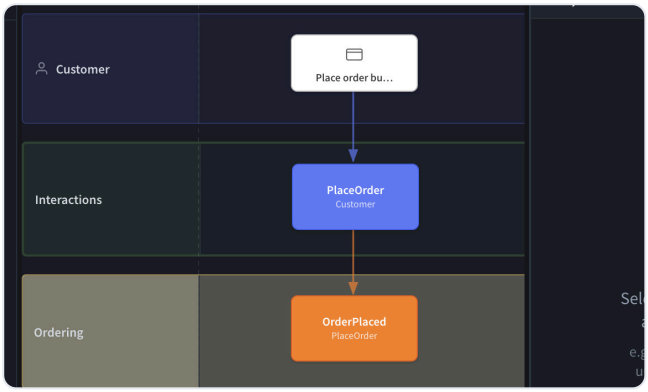


Figure 4. Process — Event Modeling: Place order button → PlaceOrder → OrderPlaced

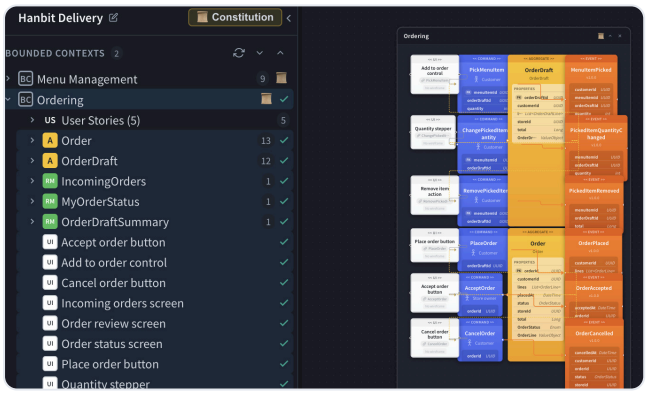


Figure 5. Design canvas — screens → commands (blue) → aggregates (yellow) → events (orange)

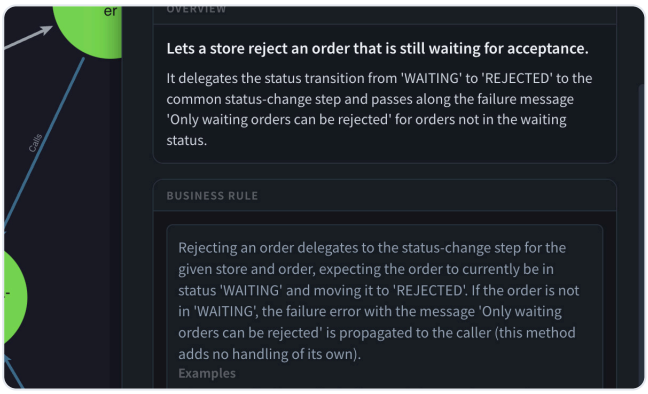


Figure 6. Legacy — code graph with each method's overview, business rules and examples

Feature cards on Home

Card	What it does	Card	What it does
Proposals	Requirement to validation	**Data Model**	Entities · value objects
Requirements	Context › feature › story	**Code**	Files · Claude Code terminal
Process	BPM · Event Modeling	**Legacy · Data Sources**	Code analysis · existing DBs
Design Canvas	EventStorming canvas	**Project Constitution**	Design principles · interview

07 · STRENGTHS

① Refines requirements into DDD and criteria

Written requirements often miss exceptions and edge cases. Claude Code follows the INTENT skill to refine a requirement into **processes, bounded contexts, features, user stories and acceptance criteria**, so you can review edge cases the sentence never mentioned.

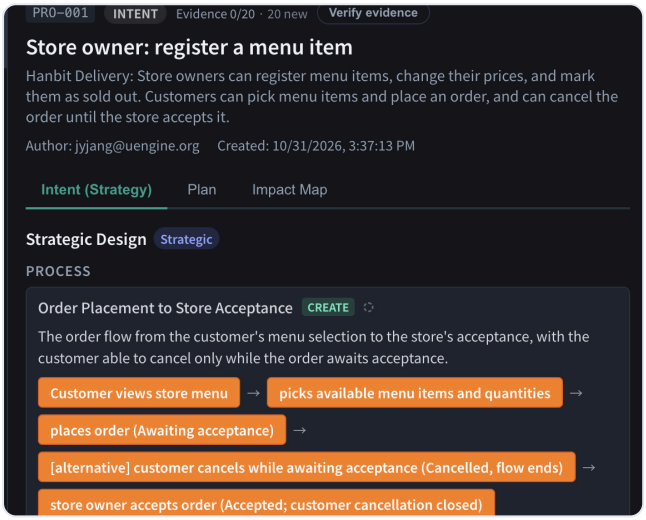


Figure 7. PRO-001 strategic design — two processes from one requirement sentence. *Demo data (tutorial exercise).*

Result (127 s)	Content
2 processes	Order Placement to Store Acceptance · Menu Upkeep
2 contexts	Ordering (Core) · Menu Management (Supporting)
7 features	Menu Item Registration, Order Placement and others
9 stories	Store owner 4 · customer 5, each with criteria

It proposes edge cases the sentence left out

- **Given** my order awaits acceptance · **When** I cancel · **Then** it is ‘Cancelled’ and the store can no longer accept it
- **Given** the store accepted my order · **When** I try to cancel · **Then** it is refused and the order stays ‘Accepted’
- **Given** a cancel and an accept arrive together · **When** both are processed · **Then** only the first takes effect

Three modes to match the scope of a change

Mode	How it works
Simplified	INTENT → PLAN in two stages, for changes with a small impact
Detailed DDD	Six stages, Discover → Decompose → … → Tactical, confirmed one by one
ODA Standard	Designed on the TM Forum ODA standards, with a conformance check

> **You can also start from a requirements document.** A pasted five-section document produced 5 contexts, 10 features, 17 user stories and 2 processes in 161 seconds.

07 · STRENGTHS

② Plans grounded in design principles

The same feature gets different plans under different stacks and architecture styles. Deciding these anew for each proposal leads to inconsistent structures, so Robo Architect keeps one **design principles document (Constitution)** per project, and every plan follows it.

Seven design principles from two questions

Question	Reason for the recommendation	Answer chosen
Architecture style	Two contexts joined at one point: a modular monolith keeps them apart at low cost	MONOLITH (recommended)
Tech stack	Order rules need transactions; Spring Boot with PostgreSQL was suggested	Typed in: Node.js (Express) · React · SQLite

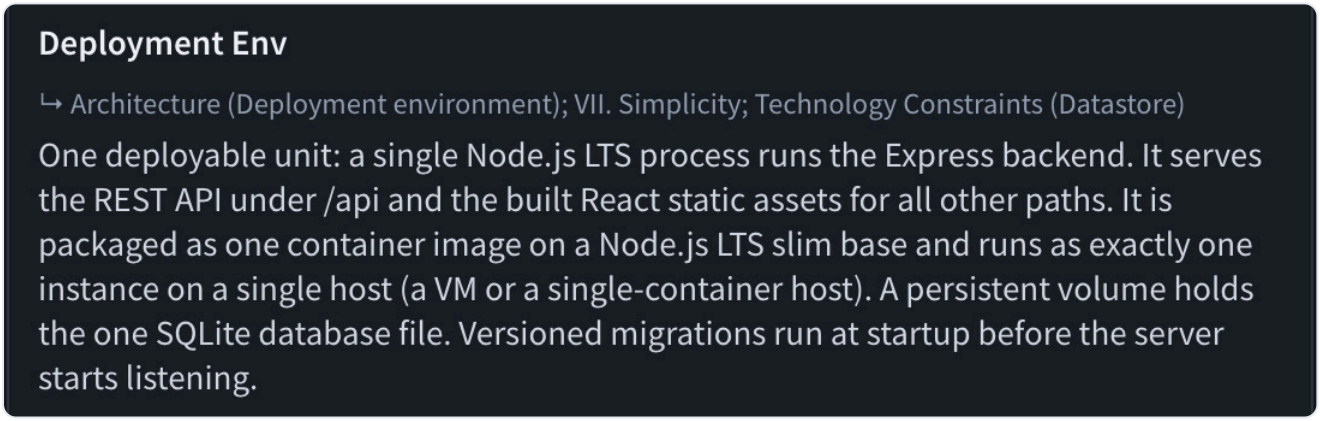


Figure 8. The Plan's decision "Deployment Env" — the principle sections it rests on (↳ Architecture; VII. Simplicity; Technology Constraints) appear with it. *Demo data (tutorial exercise).*

Decision	Content	Based on
Deployment	One Node.js process serves the API and the React app; one SQLite file	Architecture · VII. Simplicity
Data access	State re-checked in a transaction, so a conflicting accept and cancel cannot both succeed	Technology Constraints · VI
API errors	Domain errors map to explicit 400 · 403 · 404 · 409 codes	Technology Constraints

Gaps in the principles are flagged. No principle said how users sign in or who owns which store, so the plan used a development stub and marked AUTHENTICATION as a "Constitution Gap." Changing the principles marks existing plans "stale"; implementation needs a plan confirmed against the latest principles.

08 · STRENGTHS

③ Claude Code builds in isolation

AI code mixed straight into existing code is hard to review. For each proposal, Robo Architect creates a **Git worktree on a dedicated branch (`proposal/PRO-001`)**, a separate working folder of the same repository, and builds only there. The target branch is untouched until acceptance.

1. **Generate the task list** — the confirmed Plan becomes tasks by phase, each listing the files it will create.
2. **Start implementation** — a worktree and branch are created in the target project folder, and the proposal's Claude Code cell opens in the Code tab.
3. **One commit per task** — Claude Code runs `/robo-implement PRO-001`, ticking off and committing each task as it finishes.
4. **Monitor and step in** — follow the progress bar and per-phase counts; stop the work or add instructions in the cell.

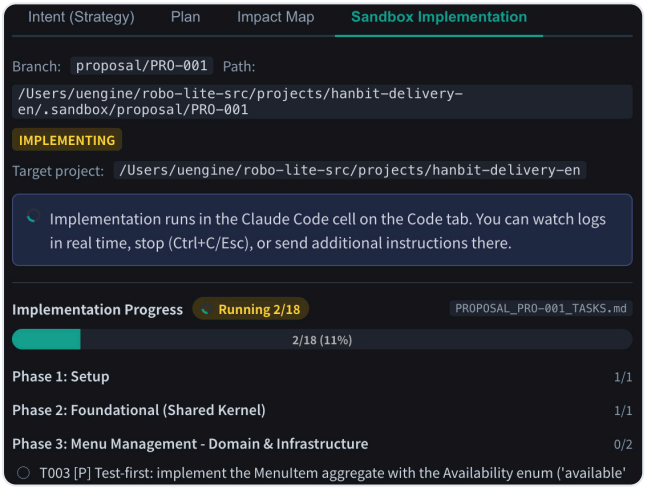


Figure 9. PRO-001 during implementation — branch, worktree path, Running 2/18. *Demo data (tutorial exercise).*

Phase	Tasks
1 Setup	Monorepo, import rules
2 Kernel	SQLite, migrations, errors
3–4 Menu	MenuItem, commands, API
5–6 Ordering	OrderDraft · Order, API
7 Frontend	Menu and order screens
8 Test	API tests, packaging

18
Tasks
8 phases

About 35 min
Build time
After the prompts

18
Commits
One per task

9,290 lines
125 files
Excluding the lock file

Folders follow the module boundaries set in the Plan

`backend/src/modules/menu-management` · `ordering` · `backend/src/shared` · `backend/migrations` · `frontend/src/features/menu` · `ordering`

You answer Claude Code's safety prompts yourself. Robo Architect does not answer folder trust or bypass-permissions consent for you. Bypass mode is meant for the cell that works only in the isolated workspace.

08 · STRENGTHS

④ Validates, then merges code and design

Finished tasks do not prove the code follows the design. **Structural validation** uses the robo-sync extractor to find the design's aggregates, fields, commands and events in the code. **Acceptance criteria validation** checks each Given · When · Then against code and tests, with evidence.



Type	Checks	PASS	FAIL	SKIP
Acceptance criteria	24	24	0	0
Structural	28	28	0	0
Total	52	52	0	0 (pass rate 100%)

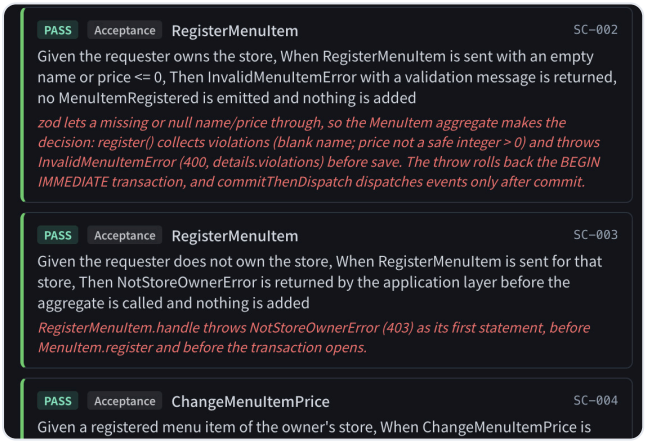


Figure 10. Acceptance criteria cards — verdict, command, scenario number and evidence. *Demo data (tutorial exercise).*

A SKIP is a check that could not be matched automatically. This run had none: every command had its own input type, so all 10 were matched field by field. SKIPs count in the pass-rate total.

A FAIL needs confirmation. Accept is enabled only after "I acknowledge the failures and will proceed with Accept," and the choice is recorded.

Acceptance and rollback

Action	What happens
Accept (Dual Merge)	Merges the branch and writes the strategic and tactical design to the graph. If the graph update fails, the merge is undone and marked as failed
Revoke (Undo)	"Graph only" or "Graph + Code" (a `git revert` of the merge)
Destroy	Discards without accepting; cleans up the workspace, keeps the history

09 · STRENGTHS

⑤ Shows a change's impact on the design

Most development changes an existing design. In a project with an accepted design, a new proposal separates **new elements (CREATE) from modified ones (MODIFY)** and shows the conflict risk where implemented elements change.



For PRO-002 we entered a new rule: a store owner who rejects an order picks one reason, and the customer sees it. INTENT (70 seconds) added the feature Store Order Rejection and two stories to the existing Ordering context, with no new context. The Plan (about 7 minutes) kept PRO-001's deployment unit, libraries and folder layout.

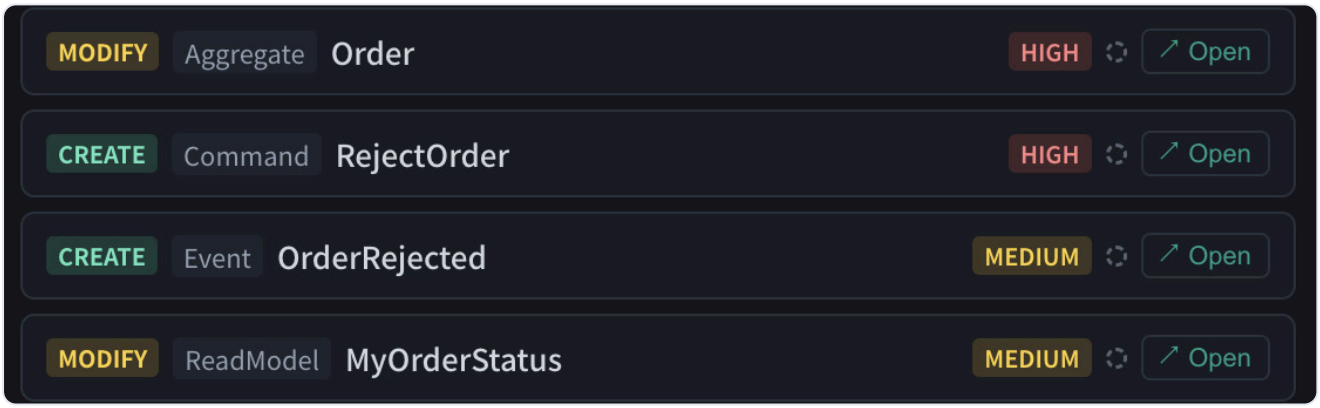


Figure 11. PRO-002's tactical design — existing Order and MyOrderStatus are MODIFY; RejectOrder and OrderRejected are CREATE. *Demo data (tutorial exercise).*

Operation	Element	Conflict risk	Change
MODIFY	Aggregate Order	**HIGH**	Rejected status, RejectionReason enum and property, 3 invariants
CREATE	Command RejectOrder	NONE	Rejects an order with one reason
CREATE	Event OrderRejected	NONE	Order rejected
MODIFY	Read model MyOrderStatus	NONE	Shows the reason to the customer

HIGH shows where to look; it does not block. Order is already implemented and accepted, so validation should confirm its existing criteria still pass. If another proposal is implementing the same element, submission to the Plan stage is blocked with a notice.

09 · STRENGTHS

⑥ Legacy code and DBs as design evidence

Business rules are often recorded more precisely in code than in documents. Robo Architect analyzes legacy code into a graph of **classes, methods, tables and business rules**, and links each element of a new proposal to the code it is based on.

1 DDL parsing · 2 Source loading · 3 Project scan · 4 Linking · 5 Context setup · 6 AI analysis · 7 Post-processing

141 s
Analysis time
Java · DDL sample system

6 · 18 · 7
Classes · methods · tables
32 columns, 46 calls

35–53
Business rules
Varies by run (35 · 52 · 53)

11 / 11
Evidence links
15 pieces of evidence

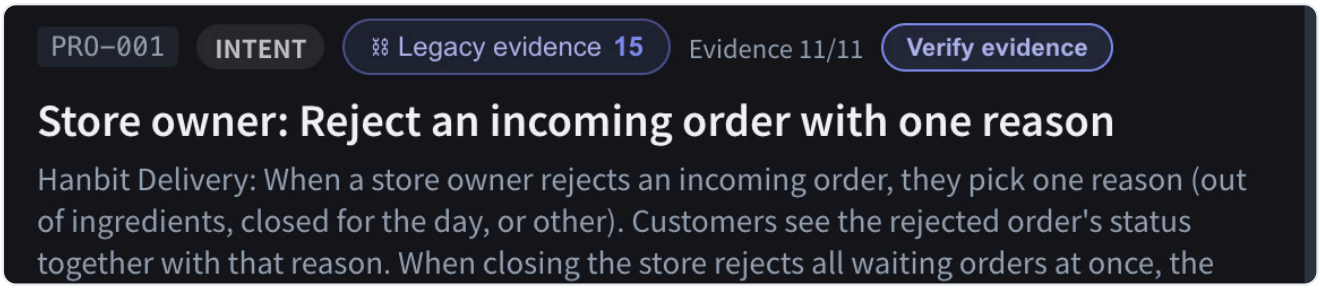


Figure 12. A proposal designed from legacy code: "Legacy evidence 15 · Evidence 11/11." *Demo data (tutorial exercise).*

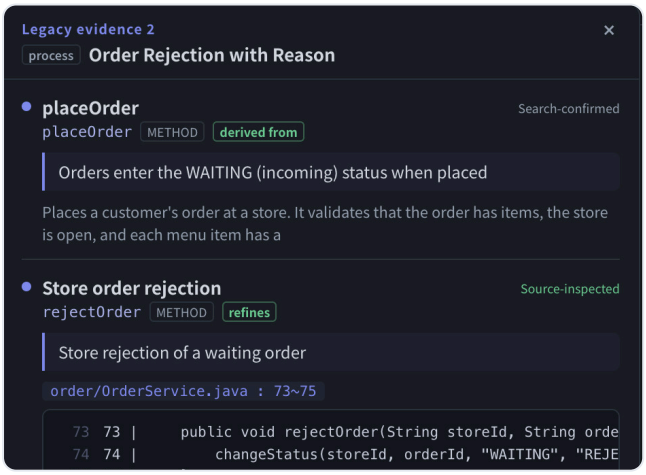


Figure 13. Evidence for the process "Order Rejection with Reason," down to the source lines. *Demo data (tutorial exercise).*

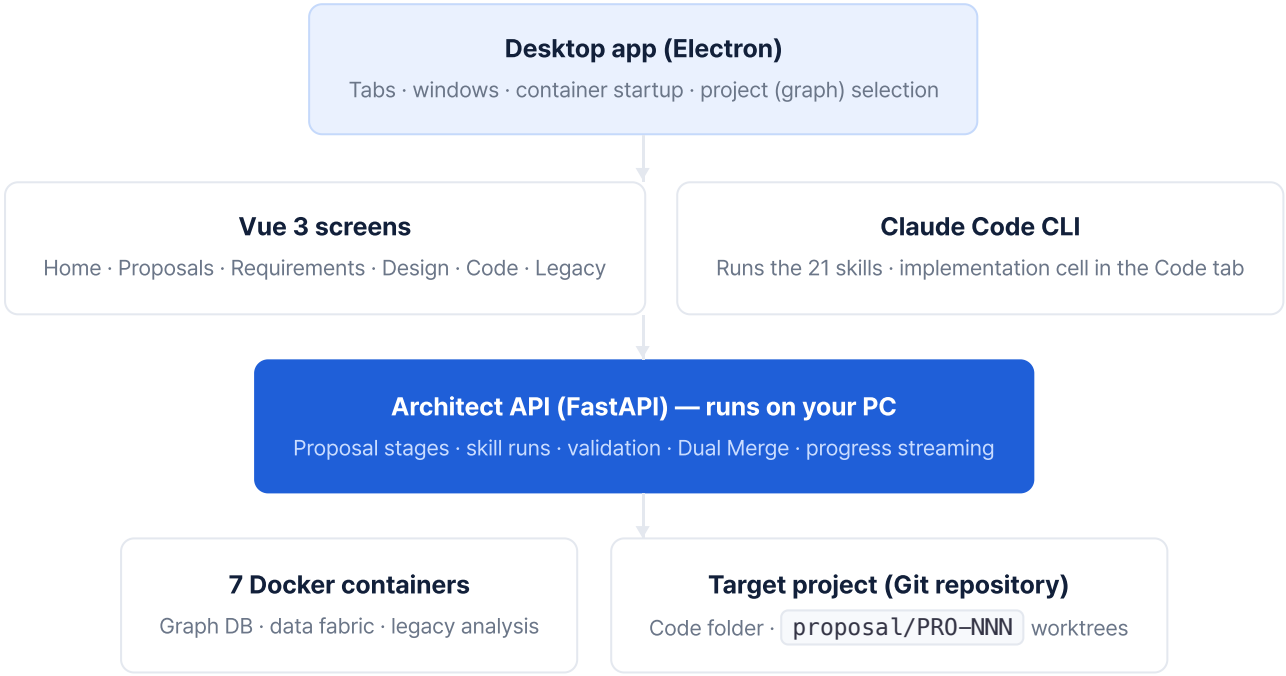
Mark	Meaning
derived from · refines	Comes from / changes old behavior
Source-inspected	Opened in the source
Search-confirmed	Seen in search results
File : line	Where the evidence points
Cited by n elements	New elements relying on it

Evidence adds conditions the requirement lacked. Because `StoreService.close` returns how many waiting orders it rejected, the design gained the criterion "I am told that 3 orders were rejected." Existing DBs can also be registered as data sources (PostgreSQL · MySQL · MariaDB).

10 · ARCHITECTURE

System architecture and runtime

Robo Architect Lite consists of **the desktop app, the Architect API running on your PC and seven Docker containers**. The app starts the containers itself and leaves them running after you quit, so the next start is faster. AI work is done by Claude Code on your PC.



Containers

Container	Group	Role
graph-db **graph-bolt**	Ontological DB	Stores the design graphs and the analysis graph, one graph per project. graph-bolt is the Bolt protocol entry point; it passes queries to the graph engine without keeping state
fabric **catalog**	Data fabric	Connects registered existing DBs to the graph DB and provides table, column and relationship information
analyzer **parser** **gateway**	Legacy analysis	Parses code (ANTLR) and analyzes its meaning and business rules. The analyzer's AI calls go through the Architect API and are handled by Claude Code

Design graphs and the analysis graph are kept separate

Legacy analysis clears the analysis graph and writes it again. The graph DB is configured to refuse to start if the analysis graph has the same name as the design graph, which prevents an analysis from erasing design data.

11 · ARCHITECTURE

Technology stack and skills

The product uses **widely adopted frameworks and open-source technology**, so your team can understand and maintain it. The AI's procedures live in skill files, apart from product code.

Layer	Components
Desktop	Electron 33 · electron-builder 25 (Windows NSIS · macOS DMG) · TypeScript
Frontend	Vue 3.5 · Vite 5 · Pinia · Vue Flow (canvas) · bpmn-js 18 · Mermaid 11 · Tailwind CSS 4 · CodeMirror 6 · xterm.js 6
Backend	Python 3.11+ · FastAPI · Uvicorn · Pydantic v2 · SSE (server-sent events) · MCP
Graph DB	Ontological DB — graph engine and Bolt gateway, Cypher queries
AI execution	Claude Code CLI (skills · Code tab cell) · OpenAI-compatible connector for the analyzer
Analysis	Legacy analyzer · ANTLR parser · data fabric · data catalog · API gateway
Testing	pytest · Playwright · ruff

21 skill files

Bundle	Skills
robo-proposals 15	Stages — `intent` · `project-constitution` · `plan` · `tasks` · `implement` · `test` · `context` (impact) Detailed DDD — `scope` · `discover` · `decompose` · `strategize` · `connect` · `define` · `tactical` Standards — `oda`
robo-spec 4	`robo-implement` (build in the Code tab cell) · `robo-sync` (code → design check) · `robo-plan` · `robo-tasks`
robo-changes 2	`robo-change-specify` · `robo-change-tasks` (requirement changes)

> **Editing a skill file changes the AI's judgment without touching product code.** We added one rule to the INTENT skill: “Append a short uppercase code in parentheses to every BoundedContext name.” Without a restart, the next analysis returned “Menu Management (MENU)” and “Ordering (ORD).”

12 · HOW TO USE

How to use it — six steps

Open a project, write a proposal, review the design and plan, then implement, validate and accept. Below are **the six steps on real product screens**, with demo data (tutorial exercise).

1 Open project · 2 Write proposal · 3 Review design · 4 Principles · plan · 5 Isolated build · 6 Validate · accept

Step 1 · Create and open a project

On Home, enter a project name (e.g. `hanbit_delivery_en`) and choose a Git repository as the code folder. **Open project** points the backend at this project's graph and enables the feature cards.

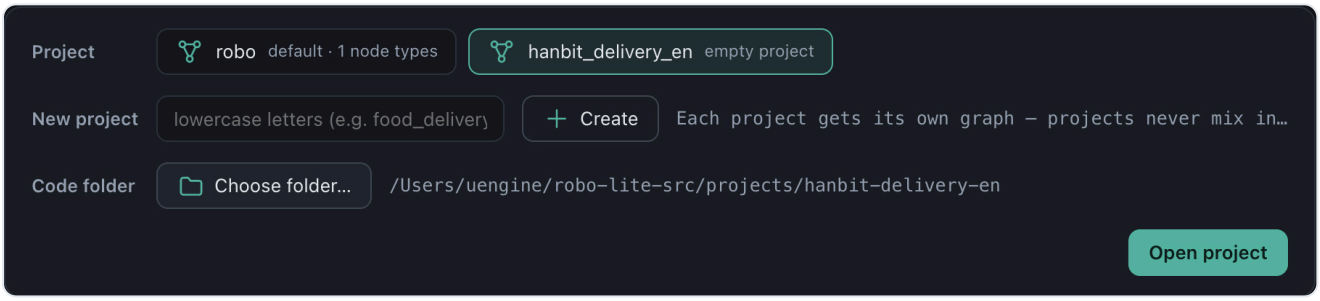


Figure 14. Project selection — each project gets its own graph, separate from all others.

Step 2 · Write a proposal

In the Proposals tab, click **+ New Proposal** and describe the change. Choose a mode (Simplified · Detailed DDD · ODA Standard) and click **Start AI Analysis**.

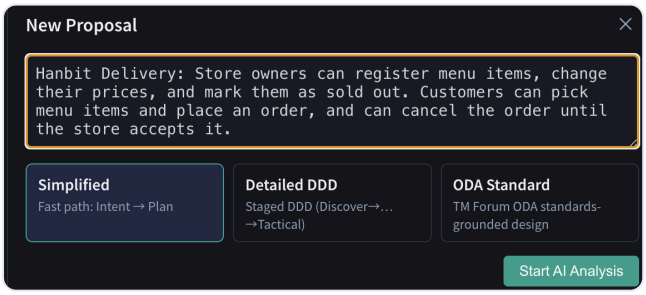


Figure 15. The New Proposal dialog

Step 3 · Review the strategic design

The progress window shows what Claude Code is doing. Then review the strategic design and Impact Map, and regenerate with feedback if needed.

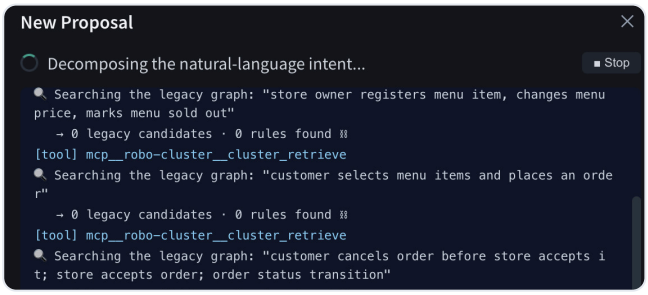


Figure 16. Analysis in progress — the legacy graph is searched for each requirement.

The AI asks when something needs clarifying

If a requirement is ambiguous, multiple-choice questions appear during the analysis. Pick an option or type an answer; to go on without answering, click **Continue with current info**.

12 · HOW TO USE

Steps 4–6 · Plan, build, validate, accept

Step 4 · Confirm principles and plan

Click **Proceed to Plan →**, then **Generate Plan**. If the project has no design principles yet, an interview opens first. Review the architecture decisions and tactical design, then click **Confirm Plan**; **Open Sandbox Implementation** appears.

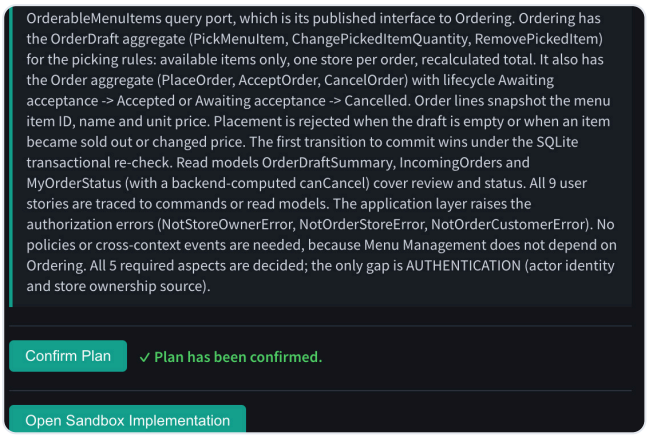


Figure 17. Plan summary and confirmation

Step 5 · Build in an isolated workspace

Click **Generate task list**, then **Start implementation**. **Go to Claude Code cell** opens the proposal's cell in the Code tab, where the work starts and you can follow file writes, tests and commits.

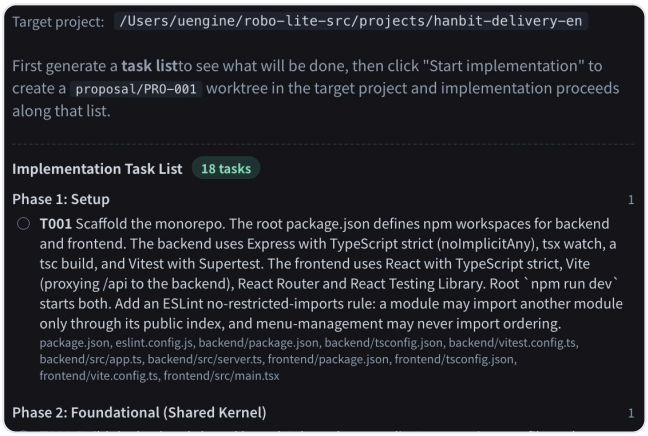


Figure 18. Task list: 18 tasks, each with its files.

Step 6 · Validate and accept

When the build is done, click **Done → Verify**. Check the structural and acceptance criteria results, then click **Accept (Dual Merge)**: code and graph DB are updated together, and the status becomes ACCEPTED.

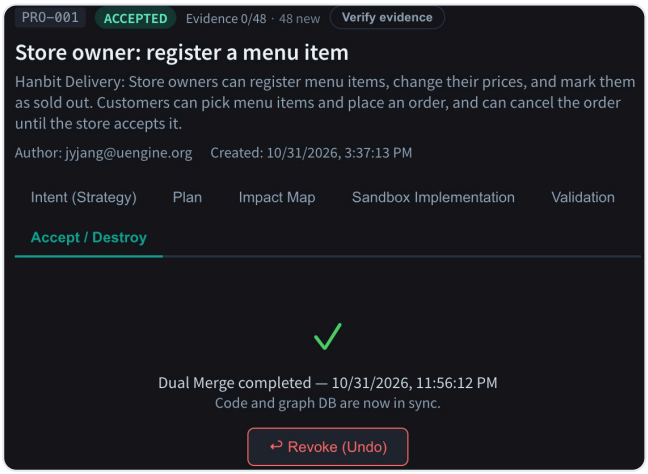


Figure 19. Dual Merge completed; **Revoke (Undo)** undoes it.

> **Stay on the tab while a long stage runs.** Reloading the tab or moving to another screen during plan generation starts it over. To do other work meanwhile, open a new tab from Home.

13 · SECURITY

Security and governance

Robo Architect Lite is a desktop app that runs on the user's PC. Before adoption, review **the scope of data processing and the change approval procedure** against the points below.

Data processing scope

- **Design data stays on the PC.** The graph DB lives in a local Docker volume, and the graph DB, Bolt and analysis ports listen only on the loopback address (127.0.0.1).
- **AI requests go through Claude Code.** It analyzes, plans and implements with the user's own login. Requests include requirements, designs and parts of the code, so check them against your Claude Code usage policy.
- **Legacy analysis calls AI without tools.** Claude Code answers the analyzer with text only, in an empty temporary folder.
- **An OpenAI API key is optional.** It is used only for document upload and chat edits in the Design tab.
- **Secrets stay out of logs.** Values such as local DB passwords go to a user-only file or the OS's encrypted store.

Change control

Item	Details
Plan check	Implementation needs a Plan confirmed against the latest design principles
Code isolation	Work happens in a branch worktree; the target branch is untouched until Accept
Conflict detection	Plan submission is blocked if another proposal is building the same element
Failure check	With any FAIL, Accept requires acknowledging the failures
Combined update	If the graph update fails, the code merge is undone and marked as failed
History · logs	Each status change records who and when; each request gets a tracking ID

AI execution settings

Proposal skills run in the Claude Code CLI by default; a setting can switch them to an OpenAI-compatible endpoint. The Code tab cell uses Claude Code.

Lite is a single-user app without accounts or permissions. Author details come from the PC's Git settings. Policies such as separate approvers become operating procedures at adoption, and network and AI policies follow your standards.

14 · INSTALLATION

Installation and delivery

Robo Architect Lite runs as one desktop app and seven Docker containers. The app loads and starts the container images itself, so **there is no Python or database to install.**

Platform	Contents	Availability
Windows 10/11	Installer (about 100 MB), image bundle (about 1.5 GB, 7 images), SHA256SUMS. Per-user install, no admin rights	Provided on request
macOS	Desktop app for Apple Silicon (DMG)	Installer being prepared for release
First run	Checks the bundle's checksum, loads it and starts the 7 containers: about 50 s on the tutorial PC, 4–5 min on an installed copy's first run	—

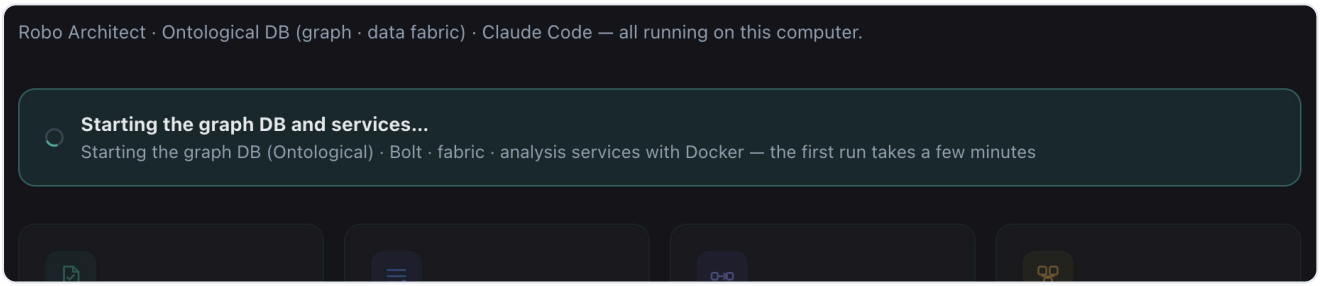


Figure 20. First run — the feature cards stay locked while the app starts the graph DB and analysis services in Docker.

Recommended environment

PC

macOS or Windows 10/11, 16 GB of memory or more, 20 GB of free disk space or more.

Docker Desktop

Keep it running while you use the app: the graph DB and services are containers.

Claude Code

Run `claude` in a terminal and sign in first. It analyzes, plans and implements.

Git

The code folder must be a Git repository; each proposal gets a worktree of it.

Deliverables stay yours. Code lands in your Git repository as per-task commits, and the design graph stays in the graph DB on the PC, where it can be queried over the Bolt protocol.

15 · QUALITY

Product development and quality assurance

The Robo Architect team follows **one process from spec to implementation and testing**. Each feature starts with a spec and a plan, and results are checked by automated tests and in the running app.

A spec-first development process

Each feature goes from spec to plan to task list to implementation, and all of these stay in the repository: currently **59 feature folders with 58 specs, 42 plans and 39 task lists**.

Automated testing

85
Backend test files
pytest · 725 test functions

72
UI test files
Playwright

14
Regression tests
No backend server needed

4
Desktop test files
Launcher · image loading

The regression suite (`npm run test:regression`) has 2 logic unit tests and 12 UI tests that replace server responses with fixed data. Static analysis uses ruff.

Usage flow verified in the real app

The 14-chapter tutorial's screens and results come from the Lite app running on macOS (Apple Silicon). Stage times:

Stage	Time	Result
INTENT (one sentence)	127 s	2 processes · 2 contexts · 9 stories
Plan generation	693 s	28 tactical items, including 3 aggregates
Implementation	About 35 min	18 tasks · 18 commits
Validation to acceptance	About 8 min	52 checks: 52 PASS · 0 FAIL · 0 SKIP
Legacy analysis	141 s	6 classes · 18 methods · 7 tables
Change proposal PRO-002	70 s + 404 s	INTENT + Plan; Order MODIFY, risk HIGH
Requirements document	161 s	5 contexts · 10 features · 17 stories

All screens in this brochure were captured from the real product during the tutorial. AI results vary slightly by run: analyses of the same code gave 35, 52 and 53 business rules, with the same class, method and table counts.

16 · REFERENCES

Track record of the base product, MSAEZ

Robo Architect builds on **experience developing and operating MSAEZ (MSA Easy)**, an EventStorming-based design and code generation tool used in industry, on a public platform and in education.

Enterprise
POSCO DX
Design and code generation for applications and microservices

Public platform
DPG Integrated Testbed
Registered and operated as a digital development tool (SaaS)

9,655
MSA School learners
325 sessions · 17,864 hours

① Industry — POSCO DX

POSCO DX uses MSAEZ to **design applications and microservices and generate their source code**, producing code from design models derived through EventStorming.

② Public sector — DPG Integrated Testbed

The DPG (Digital Platform Government) Integrated Testbed, run by Korea's National Information Society Agency (NIA), helps SMEs, startups, citizen developers and public bodies **develop, test and verify services** with public and private data, cloud resources and private-sector SaaS tools.

MSA Easy (MSAEZ) is registered there as a SaaS that **automates microservice analysis, design, implementation and deployment**.

③ Education — MSA School

MSAEZ is the **standard practice tool at MSA School**, a cloud-native development training provider. Learners design a domain with EventStorming, then generate code from the model and deploy it.

Training to date: **325 sessions, 9,655 learners, 17,864 hours**, at basic, intermediate and advanced levels, covering Biz (EventStorming · DDD), Dev (Spring Boot · Kafka) and Ops (Kubernetes · GitOps).

Area	MSAEZ (base product)	Robo Architect Lite
Design · build	EventStorming design, code from the model	DDD design per proposal, built and validated with Claude Code

From initial design to ongoing change management. On top of MSAEZ's design and code generation, Robo Architect adds change impact analysis, stage-by-stage proposals, principle-based planning, a coding agent (Claude Code) and legacy-grounded design.

* Training figures are MSA School's published totals. Scope and expected benefits are discussed individually with our sales team.

16 · ADOPTION

Adoption process and schedule

Adoption runs **in four phases**, from an environment review and a PoC (proof of concept) to full rollout. The schedule below is a standard example, adjusted to your environment and review process.



Key PoC checkpoints

- Are real requirements **refined well into bounded contexts, user stories and acceptance criteria**?
- Does the analysis of existing code **link to new proposals as design evidence**?
- Do **implementation and merging fit** your repository layout and branching practice?
- Are the structural and acceptance criteria results **enough to judge the implementation**?
- Do Claude Code usage and data handling **comply with your security policy**?

Items to agree on during adoption

AI execution

Claude Code accounts, policy and data handling scope.

Repositories

Repositories to work in and the final merge branch.

Design principles

Standards for stack, architecture and repository layout.

Skills

Your design procedures and document formats, built in.

30-minute product demo and PoC planning

We demonstrate the real product, from entering a requirement through design, planning, implementation and validation. Send us some of your business documents in advance, and **we will prepare the demo with them**.

RA

Robo Architect Lite

AI ARCHITECTURE WORKBENCH

CONTACT US

From requirement to code, manage every change together.

Contact us about adoption, a product demo or a quote. Our specialists reply within one business day, and we can also schedule a product demo for you.

COMPANY	PRODUCT	DOCUMENT
uEngine Solutions, Inc.	**Robo Architect Lite**	**Rev. 2026.10**

FOUR STAGES IN ONE PROPOSAL

Requirements
From text and documents to DDD design

Plan
Decisions based on design principles

Build
Isolated workspace · Claude Code

Validate · merge
Code and design graph updated together

uEngine Solutions, Inc.

© 2026 uEngine Solutions. Rev. 2026.10 · Figures and screens may change without notice.

uEngine