# Shipwise55

## Automating the Chief Product Officer for AI-built software.

The specification & memory layer that turns product thinking into editable text and visual system graphs.

Seed / Series A · 2026

# Vibe-coding has no CPO — so it crashes production.

## Today's tools optimize for

— Speed
— Instant demos
— Fast prototyping

## Real engineering requires

— Memory
— Architecture
— Specifications
— Long-term consistency
— Iterative reasoning

WHAT'S ACTUALLY MISSING

The Chief Product Officer role — deciding what to build, designing the architecture, owning NFRs and edge cases — is being skipped. Tools like Lovable hallucinate backends and crash in production because there is no persistent spec or system memory to reason against.

# Bigger context windows alone don't solve it.

### The industry response

— Larger context windows
— Brute-force token scaling
— More expensive inference

### What a useful system must do

— Know what matters
— Remember past decisions
— Retrieve exact architecture state
— Localize changes
— Reason hierarchically
— Preserve consistency over time

**The real bottleneck is not token count.**
**The real bottleneck is persistent, structured memory.**

# The missing layer in AI software engineering.

## Current products

Lovable, Cursor, Replit, Devin, Windsurf — all focus on:

— Generating code quickly
— Compressing developer workflows
— Accelerating implementation

## What none solve

— Persistent architectural memory
— Specification-driven engineering
— Long-context orchestration
— Modular system reasoning
— Enterprise-grade maintainability

THE AI SOFTWARE STACK

**Foundation models**
↓ **Coding agents**
↓ **Shipwise55 memory layer**
↓ **Enterprise software systems**

# Automate the CPO. Make software maintainable in text and graphs.

FROM

**"Prompt an LLM, hope the backend doesn't crash."**

TO

"Edit the spec or the system graph

in plain English — only affected modules regenerate."

## Shipwise55 automates the CPO

- — Requirements extraction from PRDs, transcripts, ideas
- — Architecture and service decomposition
- — Non-functional requirements and constraints
- — Edge cases and dependency mapping
- — Spec-first validation before generation

## Editable internals — text + graph

- — Every decision stored as human-readable text
- — Every relationship visible as a navigable graph
- — Localized edits — change one node, regen one module
- — No full-context rewrites, no architectural drift
- — Maintainable production software, not throwaway demos

# The Shipwise55 workflow.

**STEP 01 — UPLOAD**

Product idea, PRD, meeting notes, whiteboards, architecture docs, business requirements.

**STEP 02 — EXTRACT**

Use cases, actors, requirements, permissions, entities, APIs, workflows, dependencies, edge cases.

**STEP 03 — GENERATE**

Specifications, architecture maps, system diagrams, UI mockups, service decomposition, implementation roadmap.

**STEP 04 — REVIEW**

User reviews and edits specifications. Architecture validated before any code is generated.

**STEP 05 — BUILD**

AI coding agents generate modular implementations against the validated specification.

**STEP 06 — PERSIST**

A living memory layer tracks system evolution. Future edits localize to affected modules.

# Specifications are cheaper than code.

## Why traditional engineering works

— Humans review requirements before implementation
— Architecture decisions happen before coding
— Stakeholders understand system behavior

## Modern AI coding skips this

— Uncontrolled generation
— Expensive rework
— Architecture instability
— Hallucinated dependencies
— Fragile systems

## Shipwise55 restores

— Architectural reasoning
— Human-readable system understanding
— Iterative requirement refinement
— Localized system updates

**Changing specifications is dramatically cheaper than rewriting generated code.**

# The Hierarchical Memory Transformer.

### Not just

— Pure sparse attention
— Pure RAG
— Pure LongRoPE scaling
— Brute-force context windows

### Instead — five cooperating layers

— Local transformer reasoning
— Learned long-term memory
— Sparse block routing
— Semantic KV-cache compression
— Exact retrieval fallback

## Massive effective context windows — with architectural consistency.

# How we solve the context window problem.

**01**

## Local active reasoning

High-quality short-range coding reasoning preserves code quality and instruction following.

**02**

## Specification memory

Persistent understanding of product requirements across sessions.

**03**

## Architecture graph memory

Relationships between services, APIs, flows, and dependencies tracked as a living map.

**04**

## Sparse retrieval routing

Selective retrieval of only relevant architectural context — slashing token cost and hallucinations.

**05**

## Exact source retrieval

Precise retrieval of implementation details and prior decisions when consistency matters.

# The production crash problem — solved at the root.

### What killed vibe-coding in production

— Lovable-style hallucinated backends and crashes
— Models forget prior decisions every session
— Codebases drift and become unstable
— Small changes trigger cascading regressions
— Non-technical owners lose all visibility

### What Shipwise55 actually delivers

— An always-current spec in plain English
— An editable system graph of services, APIs, flows
— Localized regeneration — change one node, ship one module
— Architecture-aware coding agents that don't drift
— Production-grade software you can maintain, not just demo

**The bottleneck wasn't the model. It was the absence of a CPO and a persistent system memory. Shipwise55 supplies both.**

# Documentation as the source of truth. Code as a side-effect.

## Today — Lovable, Replit, vibe-coding

— Prompt → code, directly, with no internals exposed
— Non-developers cannot see what's under the hood
— Small amendments overload the context window
— Every fix introduces new bugs
— Backends hallucinate and crash past a threshold
— No shared artifact between stakeholders and AI

## Tomorrow — Shipwise55

— LLMs build comprehensive system documentation from your idea
— Every actor, flow, API, entity, edge case in plain English
— Review and amend a sentence before any tokens are spent on code
— Edits are localized to affected modules — never the whole project
— Code runs against a stable architecture, not a fragile prompt history
— Founders, business stakeholders, and AI read the same document

THE ECONOMIC ARGUMENT

A spec is dramatically cheaper than code — for humans and LLMs. Reviewing what needs to be built before shipping saves man-hours and tokens. Localized edits make code changes immune to hallucinations and context-window limits.

# Existing AI coding tools are stateless.

| COMPANY | MAIN FOCUS | LIMITATION |
|---|---|---|
| **Lovable** | Rapid generation | No persistent architecture memory |
| **Cursor** | Developer productivity | Limited long-term context consistency |
| **Replit** | Fast prototyping | Architecture drift over time |
| **Devin** | Autonomous execution | Weak persistent system reasoning |
| **Windsurf** | Coding acceleration | Prompt-centric workflows |
| **Shipwise55** | Persistent memory, specification-first workflows | Architecture consistency, modular reasoning, enterprise scale |

**"Cursor helps developers write code. Shipwise55 helps organizations think, remember, and evolve software systems with AI."**

# Deep infrastructure moat.

## Shipwise55 IP

— Proprietary hierarchical memory architecture
— Long-context orchestration engine
— Persistent architecture graph
— Specification decomposition engine
— AI-native system memory layer
— Long-context evaluation datasets
— Modular AI execution framework
— Enterprise architecture consistency engine

## Competitors optimize for

— UI
— Generation speed
— Wrapper experiences

## We optimize for

— Memory systems
— Architectural reasoning
— Persistent state
— Scalable software evolution

# Timing is perfect.

**1**

LLMs are now capable enough for meaningful software generation.

**2**

Context limitations are becoming the dominant bottleneck.

**3**

Enterprises need maintainability, governance, and architectural consistency.

**The next generation of AI infrastructure will be memory-native, architecture-aware, long-context optimized, and persistent by default. Shipwise55 sits at the center of that transition.**

# A massive emerging market.

## Markets impacted

— AI coding tools
— Enterprise software engineering
— AI developer infrastructure
— Product management platforms
— Enterprise architecture systems
— AI agent infrastructure
— Long-context AI orchestration

## Every large enterprise will need

— Persistent AI memory
— Architecture-aware reasoning
— Modular AI orchestration
— Long-term AI system consistency

# Multi-layer revenue model.

**SaaS subscriptions**
Founders, startups, product teams.

**Enterprise platform licenses**
Large engineering organizations.

**Usage-based orchestration**
AI generation and memory compute.

**API infrastructure**
Persistent memory APIs for AI agents.

**Private enterprise deployment**
Security-sensitive organizations.

**Architecture memory cloud**
Long-term persistent system state.

**Future potential:** **AI-native enterprise operating systems.**

# Phased market expansion.

### Phase 1 — Founders & indie builders

AI-generated apps become unstable quickly. We provide structured specifications and stable modular generation.

### Phase 2 — Product teams & CTOs

Scaling AI-generated systems across teams. We provide persistent architecture and maintainability.

### Phase 3 — Enterprise engineering organizations

Governance, consistency, compliance, architecture drift. We provide long-context orchestration and AI system memory.

### Phase 4 — AI-native enterprises

Shipwise55 becomes the infrastructure for autonomous software evolution.

# Key technical metrics.

## We aim to demonstrate

- Reduced hallucinations
- Lower regression rates
- Reduced token consumption
- Better architecture consistency
- Better long-session memory retention
- Improved large-repo task completion
- Reduced engineering iteration cost
- Higher modular code stability
- Better dependency consistency

## Benchmark categories

- Large repository modification
- Architecture drift testing
- Multi-session consistency
- Long-running agent memory
- Dependency resolution accuracy
- Enterprise workflow orchestration

# The path to AI-native infrastructure.

- **2026** Specification-first AI engineering platform.
- **2027** Persistent memory architecture engine.
- **2028** Long-context autonomous coding agents.
- **2029** Enterprise AI-native software infrastructure.
- **Future** Self-maintaining enterprise software systems.

## Long-term vision

— AI-native ERP generation
— Autonomous architecture evolution
— Organizational software memory
— Enterprise digital twins
— Persistent AI engineering organizations

# Founders who can get from 0 to 1 — and keep the runway long.

## Anton Kravchenko — Founder & CEO

— Fintech and Web3 infrastructure expertise
— Complex trading and financial systems
— Deep systems thinking, product and architecture focus
— Repeat 0 to 1 builder with next-round fundraising experience

## How we operate

— Balanced R&D and revenue-driving product surfaces from day one
— Cash burn held under a managed cap — runway measured in years
— Milestone-driven fundraising, not narrative-driven
— Distributed company with offline bases in Lisbon & New York
— Reachable in person, by mail, or by phone — not just Discord

REACH ANTON DIRECTLY

**WhatsApp: +351 932 968 173**

wa.me/351932968173

TALENT PIPELINE — SOLVING THE MEMORY PROBLEM AT THE SOURCE

**Direct access to the MIPT (mipt.ru) community — including young researchers pursuing PhDs in neural networks who can join the team to attack LLM memory at the architecture level.**

# The future of software engineering is persistent AI memory.

The current generation of AI tools helps humans write code. The next generation of AI infrastructure will:

- Remember systems
- Evolve architecture
- Preserve organizational knowledge
- Orchestrate modular reasoning
- Maintain software autonomously

SHIPWISE55 IS BUILDING

**The memory architecture layer for AI-built software.**

AND ULTIMATELY

**The operating system for autonomous software engineering.**

# Software engineering needs memory, structure, and hierarchical reasoning.

Current AI coding tools generate code. Shipwise55 enables AI to:

- Think architecturally
- Remember persistently
- Evolve systems safely
- Reason hierarchically
- Scale software generation reliably

## The future of software engineering is not prompt-to-code.
## It is persistent AI system memory.