WIX Engineering

Why I Stop Prompting and Start Logging: The Design-Log Methodology

Yoav Abrahami

Jan 2026

About Me @ Wix

Years at Wix: 16 (20) years

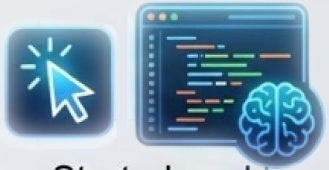
CTO, Wix Enterprise
(past: Head of Dev Platform, Chief Architect)

Building

- Future of the Web (code name Web 5.0)
- Jay Framework (Design to Code)

yoavabrahami

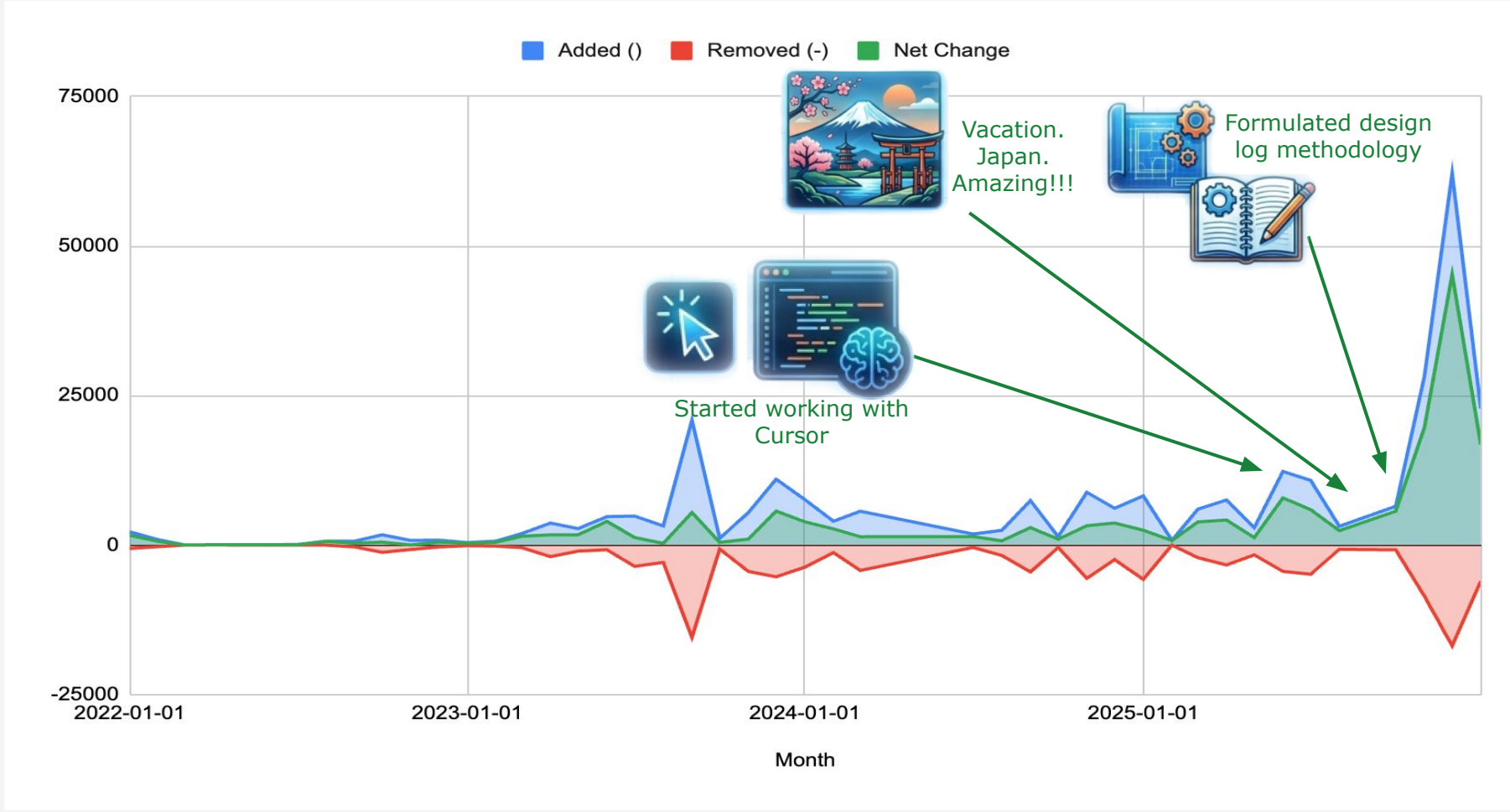
@yoavabrahami



Started working with Cursor

Vacation. Japan. Amazing!!!

Formulated design log methodology


Added ()
Removed (-)
Net Change
75000
50000
25000
0
-25000
2022-01-01
2023-01-01
2024-01-01
2025-01-01
Month
Vacation.
Japan.
Amazing!!!
Formulated design
log methodology
Started working with
Cursor

The Problem – The "Context Wall"

AI starts off great on new projects, but as codebases grow, it loses the plot.

The "Velocity Trap": Moving fast at first, then slowing down to fix AI hallucinations.

Conflicts: AI suggests code that violates previous architectural decisions.

The burden of "re-explaining": your project context in every new chat.

Let's be honest, it happens to us humans as well as a project grows

Introducing the Design-Log Methodology

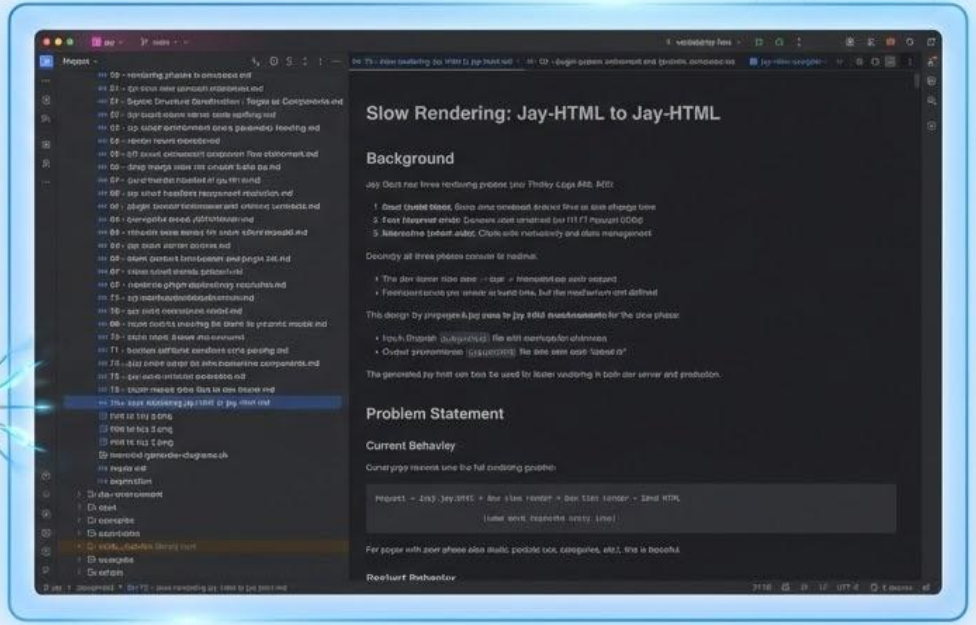
The Concept: Stop treating AI as just a “coder” and start treating it as an “**architectural partner.**”

What is a Design Log?

A version-controlled folder: **./design-log/**.

Markdown files representing a "**snapshot in time.**"

The "**Brain" of the project**: AI reads the logs before it touches the code.



Documentation vs. Design Logs

But isn't just another way to create documentation?

Traditional Docs/READMEs

Try to stay "current" but often become outdated and misleading.

Design Logs

Immutable history. They record why a decision was made at a specific moment.

Key Insight

AI and Humans needs to understand the evolution of the logic, not just the final state.

The 5-Step Workflow

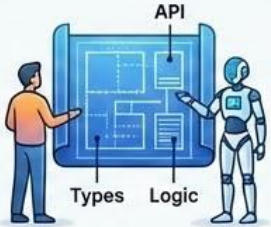
1. The Simple Prompt

Give a high-level intent (e.g., “Add Server Actions”).

2. AI Discovery

The AI creates a log (e.g., Design Log #63) and asks clarifying questions.

3. The Blueprint



You and the AI finalize the API, types, and logic in Markdown (English) before coding (Typescript in my case).

4. Zero-Drift Implementation

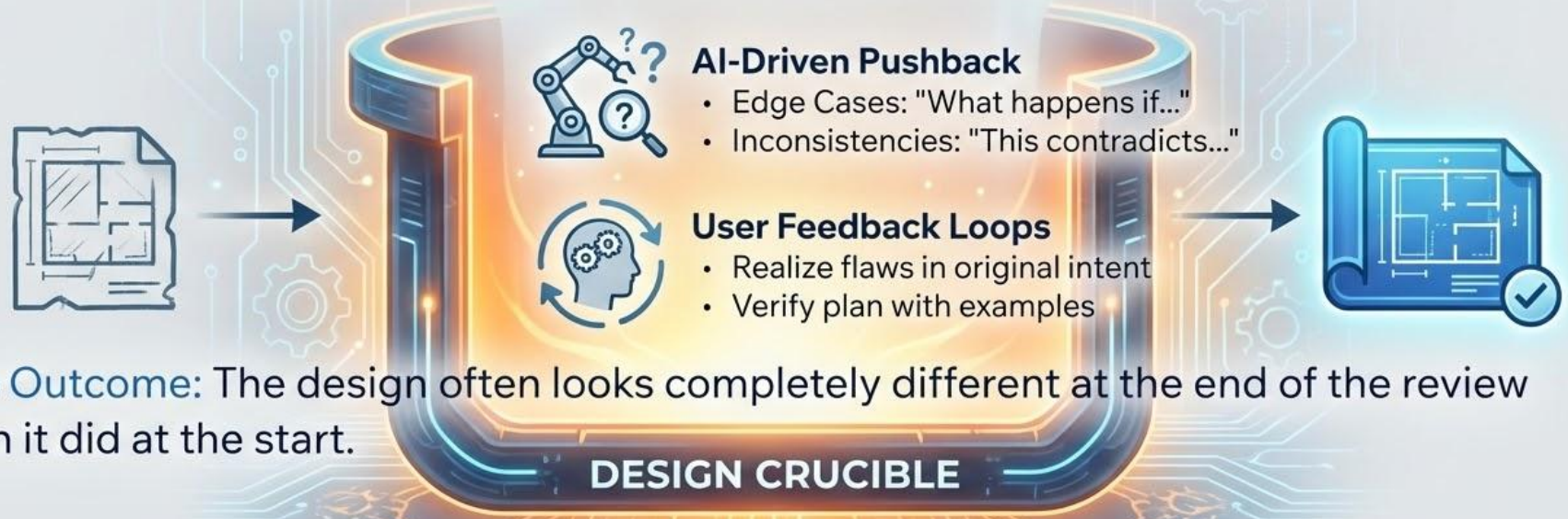
The AI follows the plan. Any deviations are recorded in the “Implementation Results" section.

5. Human Feedback & Append

Human reviews the implementation; AI updates the code and appends the findings to the Design Log.

The Critical Review Step

The "Crucible" of Design: This is the most important part of the lifecycle.



The Outcome: The design often looks completely different at the end of the review than it did at the start.

Wix Data Plugin Design Evolution

Example

Original Design:

- Generic contracts with `fields[]` array for dynamic data
- Factory functions create component instances per collection
- Configuration maps schema fields to viewState slots (title, description, image)

Feedback:

- Jay supports dynamic contracts (Design Log #60) - contracts generated at build time
- Wix Data APIs: `listDataCollections()`, `getDataCollection()` provide full schema
- Field mappings redundant if contract matches schema directly

Revised Design:

- Dynamic contracts generated from Wix Data schema at build time
- Shared components receive contract via `DYNAMIC_CONTRACT_SERVICE`
- Config simplified to routing (slugField, pathPrefix) and relationships only
- Contract tags match collection fields exactly - no mapping layer needed

The Economics of Design Logs

Q&A Section

The AI identifies edge cases and missing information (e.g., “How do we handle loading states?”) and waits for your answer.

Avoid the “Frankenstein” Codebase

Context Drift Trap

Realizing a design flaw mid-implementation leads to the **“Context Drift” trap**, where AI patches over errors with “zombie logic” instead of refactoring. This pollutes the AI's working memory and quickly throws the project off the rails.

The “Stop, Refine, and Restart” Advantage

The Design-Log methodology forces real-world pushback into a “cheap” conceptual phase. A **“Stop, Refine, and Restart”** approach minimizes implementation risk.

Recovering from the Implementation Wall

Design Log #49: full stack component rendering manifest

The Plan: A complex "Full Stack Manifest" based on TS mapped types.

The Crash: During coding, TS mapped types limitations throw the AI into running in circles that do not converge.

The Logged Exit: Instead of forcing the code to work, the team stopped. They documented the failure, effectively "closing the case" on that approach.

Documented Failure = Lesson

Insight from Failure

Design Log #50: jay stack - headless configuration

The "Clean Slate": Using the failure of #49 as a requirements list, Log #50 was created to tackle a "Headless Configuration".

The Win: Because the failure was documented, the AI "understood" the pitfalls to avoid. New approach, new plan, clean implementation.

The System Rules (The "Secret Sauce")

To make this work, the AI must follow four strict rules:

Read Before You Write:
Check existing logs for context.

Ask Questions:
For anything that is not clear, or missing information

Design Before You Implement:
No code without an approved log.

Immutable History:
Once implementation starts, the 'Design' section is frozen.

Traceable Implementation:
Document every deviation and test result.

The Rule!

Jay Framework Project Rules

Design Log Methodology

The project follows a rigorous design log methodology for all significant features and architectural changes.

Before Making Changes
1. **Check design logs** in `./design-log/` for existing designs and implementation notes
2. **For new features**: Create design log first, get approval, then implement
3. **Read related design logs** to understand context and constraints

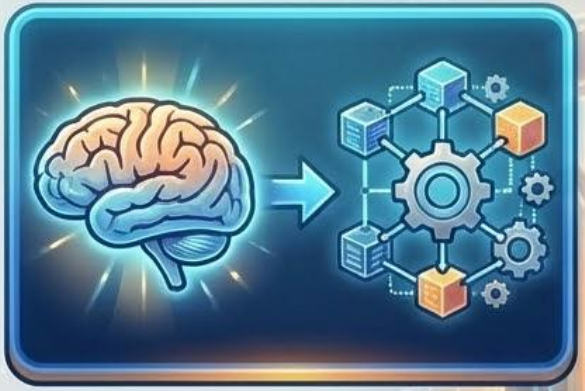
When Creating Design Logs
1. **Structure**: Background → Problem → Questions and Answers → Design → Implementation Plan → Examples → Trade-offs
2. **Be specific**: Include file paths, type signatures, validation rules
3. **Show examples**: Use ✅/❌ for good/bad patterns, include realistic code
4. **Explain why**: Don't just describe what, explain rationale and trade-offs
5. **Ask Questions (in the file)**: For anything that is not clear, or missing information
6. **When answering question**: keep the questions, just add answers
7. **Be brief**: write short explanations and only what most relevant
8. **Draw Diagrams**: Use mermain inline diagrams when it makes sense
9. **Define verification criteria**: how do we know the implementation solves the original problem

When Implementing
1. **Follow the implementation plan** phases from the design log
2. **Write tests first** or update existing tests to match new behavior
3. **Do not Update design log** initial section once implementation started
4. **Append design log** with "Implementation Results" section as you go
5. **Document deviations**: Explain why implementation differs from design
6. **Run tests**: Include test results (X/Y passing) in implementation notes
7. **After Implementation** add a summary of deviations from original design

When Answering Questions
1. **Reference design logs** by number when relevant (e.g., "See Design Log #50")
2. **Use codebase terminology**: ViewState, Contract, JayContract, phase annotations
3. **Show type signatures**: This is a TypeScript project with heavy type usage
4. **Consider backward compatibility**: Default to non-breaking changes

Summary & Key Takeaways

Shift your mindset:



Move from ‘Prompt Engineering’ to ‘Context Engineering.’’

Write the history together:

Use Markdown as the bridge between human intent and AI execution.

Predictability > Speed:

The Design-Log Methodology makes AI development stable and scalable.

WIX Engineering

Thank You.

Let's Talk!

yoav@wix.com

"Generated by AI. **Fact-checked by a very tired human.**"

Yoav Abrahami

Jan 2026