The High Cost of Broken Flow

- Modern development is a battle against interruption.
- Context switching between UX design, boilerplate code, API integration, and dependency management constantly pulls us out of our most productive state.
- This friction is not a personal failure; it’s a systemic problem. We need a methodology designed to protect our most valuable asset: **Flow State**.

A Methodology for Preserving Flow

> “Iterative Flow Development (IFD) is a methodology for rapid software development using AI assistance while maintaining engineering rigour. Its single-minded focus is **preserving developer flow state.**”

IFD structures the development process to minimise context switching and leverage AI for repetitive tasks, allowing the developer to focus on UX, architecture, and creative problem-solving.

The Core Philosophy: Focus, Flexibility, and Freedom

Flow State Preservation

Minimise context switching. The developer focuses on UX and architecture; the LLM handles the boilerplate. Work in focused 2-3 hour sessions.

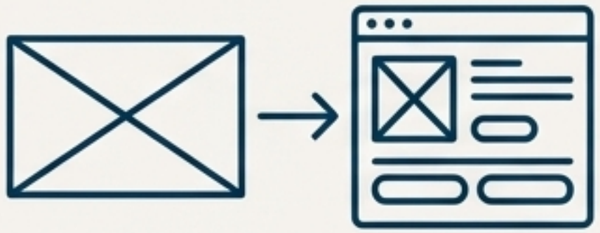
Dual-Mode Flexibility

Adapt your process with two distinct modes: **Vibe Coding** for rapid, AI-driven prototyping and **Air-Gapped Mode** for deliberate, human-reviewed production development.

Zero External Dependencies

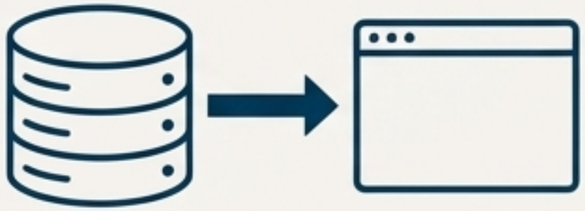
Build on the native web platform (ES6+, Web Components). No frameworks like React or Vue. This ensures longevity, simplicity, and performance.

The Core Disciplines: How We Build



UX-First Development

Define the user experience *before* writing code. Describe the intended UX to the LLM and iterate based on real user interaction, not abstract technical goals.



Real Data From Day One

No mocked data or stubbed services. The backend API must exist and be used from version `v0.1.0`. This catches integration issues immediately.

Progressive Enhancement

Start with the simplest possible version and add features incrementally. For example: `v0.2.1` (basic functionality) → `v0.2.2` (input validation) → `v0.2.3` (error handling).

Versioning: The Engine of Iteration and Isolation

Minor Versions (e.g., v1.3.1 → v1.3.2):
Are co-dependent and **share code**. Represent incremental feature development.
Rule: `v0.2.4` can depend on `v0.2.2`, but not `v0.1.5`.

Release Versions (e.g., v1.0.0, v4.0.0): A snapshot of the previous major version. No code changes, only documentation updates.

Major Versions (e.g., v1.1.0, v2.5.0): Are completely isolated. They **do not share code**. Consolidate proven features from the previous minor version series.
Rule: To reuse code, you **copy**, don't link.

The IFD Workflow: From Idea to Feature in Four Steps

1. Start with UX Description

Clearly articulate the desired user experience and functionality.

2. LLM Generates Initial Code

The AI generates the component structure, HTML, CSS, and event handlers.

3. Human Refines

Review, integrate, and test the AI-generated code against the real backend. Handle edge cases. This is where `Air-Gapped vs Vibe Mode` applies.

4. Progressive Improvement

Commit the change as a new minor version and repeat the cycle for the next enhancement.

Critical Anti-Patterns to Avoid

- ❌ **Don’t** create shared code between Major versions.
- ❌ **Don’t** use external JavaScript frameworks or libraries.
- ❌ **Don’t** mock data or stub APIs.
- ❌ **Don’t** over-engineer in v0.1.0. Keep it minimal.
- ❌ **Don’t** add business logic to schema files.
- ❌ **Don’t** break the air gap in Air-Gapped Mode.

Moving to Production: The v1.0.0 Readiness Checklist

Code Quality

- Consistent error handling throughout
- Memory management (cleanup in `disconnectedCallback`)
- No console errors or warnings

Architecture

- Clear separation of concerns
- Event-driven component communication (`CustomEvents`)
- Components are self-contained

Performance

- Lazy loading for heavy components
- Debouncing for rapid actions
- Caching for expensive operations

Maintainability

- Intuitive file organisation
- Documented component APIs
- Consistent naming conventions

The Payoff: Confidence, Freedom, and Speed

Confidence From Day One

Because you use real data from `v0.1.0`.

When `v1.0.0` ships, you know it works because you've been testing with the actual backend since the first hour of development.

Freedom to Experiment

Because version independence means no accumulated technical debt.

If `v0.4.5` is a dead end, simply abandon it and build `v0.4.6` from `v0.4.4`. Experiment without fear of breaking the main application.

Unprecedented Speed

Because you stay in flow.

With 15-30 minute iteration cycles and the LLM handling boilerplate, you can deliver a working, potentially shippable version in a single 2-3 hour session.

IFD at a Glance: The Cheat Sheet

Aspect	IFD Approach
Dependencies	None (native web only)
Components	Web Components (custom elements)
Communication	`CustomEvents` (event-driven)
State	Component-local + event coordination
Versions	Independent, no shared code (copy, don’t share)
API	Real from `v0.1.0`, no mocks
Testing	Manual + real data continuously
AI Role	Code generation + refinement
Human Role	Architecture + UX + decisions

Your Focus is the Asset. Protect It.

"IFD is about maintaining your flow state. The methodology exists to keep you focused on solving user problems, not fighting with tools or technical debt."

"Let the AI handle the boilerplate, you handle the creativity and decision-making."