Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

"Make it Pop, but not Like That": A Taxonomy of Iterative Prompting Strategies for Refining AI-Generated Web Interfaces

Zhenjiang Song

admin@ytecn.com

Anyang Preschool Teachers Colleg

Research Article

Keywords:

Posted Date: March 8th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-8994174/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

Abstract

The rapid proliferation of Large Language Models (LLMs) and generative tools (e.g., GPT-4, Tongyi Lingma, Trae) has fundamentally democratized the landscape of web development, shifting the paradigm from manual syntax construction to natural language intent specification. However, while the barrier to "drafting" initial code has been lowered, a significant "Refinement Crisis" has emerged. As task complexity scales from static landing pages to dynamic, stateful applications, novice users encounter a profound "Gulf of Evaluation" when attempting to repair AI-generated errors. Unlike text generation, where errors are semantic and visible, web interface generation involves a complex interplay between visual presentation (CSS) and invisible state management (JavaScript). In this paper, we present a large-scale observational study with **200 novice participants** tasked with utilizing IDE-integrated AI assistants to build a fully functional CRUD (Create, Read, Update, Delete) note-taking application.

Through a rigorous analysis of interaction logs and source code snapshots, we reveal that while 90% of users could generate a baseline prototype, **80% encountered severe "invisible state" breakdowns** (e.g., data persistence failure), and **50% suffered from persistent layout regressions**. We contribute a detailed taxonomy of four repair strategies: *Perceptual Refinement*, *Behavioral Correction*, *Diagnostic Proxy*, and *Global Reset*. Furthermore, we characterize the **"Whack-a-Mole" effect**—a phenomenon where repairing visual elements inadvertently corrupts functional logic due to the AI's lack of holistic state awareness. Our findings provide empirical evidence for the limitations of current chat-based coding interfaces and offer critical design implications for future "State-Aware" AI IDEs that reify invisible data flows to bridge the gap between user intent and execution.

1. Introduction

1.1 The Paradigm Shift: From Syntax to Intent

The landscape of software engineering is undergoing a seismic shift. For decades, the primary barrier to entry was **Syntax**: learning the rigid grammar of languages like JavaScript or Python. Today, Large Language Models (LLMs) such as GPT-4, Tongyi Lingma, and Trae have effectively democratized code generation, transitioning the user's role from a "Writer of Code" to an "Architect of Intent." However, this lowering of the floor has inadvertently raised the ceiling for **Validation**. In the context of web development, the output of an LLM is not a single linear text but a complex, multi-dimensional artifact comprising **Structure (HTML)**, **Presentation (CSS)**, and **Behavior (JavaScript)**. While a novice user can easily prompt *"Make a note-taking app,"* they often lack the mental model to understand the invisible dependencies between these layers. When the generated application fails—for instance, when a note disappears after a page refresh—the user is thrust into a **"Refinement Crisis."** They must debug a system they did not write and do not understand, often using vague natural language that the AI misinterprets.

1.2 The Unique Challenge of Web State

Unlike text generation tasks (e.g., writing an email), where errors are semantic and visible (e.g., a wrong tone), errors in web generation are often **State-Based and Invisible**. Our preliminary observations indicate a critical misalignment:

The Desktop Metaphor

Novice users often transfer their experience from desktop applications (like Word or Excel) to the Web. They assume that if they "edit" a text on screen, it is saved; or if they "open" a new page, the data travels with them.

The Stateless Reality

The Web is inherently stateless. Without explicit instruction to use localStorage or a database, any data entered into the DOM is ephemeral. This gap creates a profound **"Gulf of Evaluation"** (Norman, 1986). The user sees the visual interface (the "View") but is blind to the underlying data flow (the "Model"). When they attempt to repair these issues using visual descriptors (e.g., *"Make the button red"*), they often inadvertently trigger regressive bugs in the logical layer, a phenomenon we term the **"Whack-a-Mole" Effect**.

1.3 Research Context and Contributions

To empirically investigate this phenomenon, we orchestrated a large-scale observational study with **N = 200 novice developers** in a controlled university setting. The participants, who had zero prior coding experience, were tasked with building a fully functional **CRUD (Create, Read, Update, Delete)** application using AI tools. This paper moves beyond simple success/failure metrics to analyze the *process* of failure. We contribute:

A **Taxonomy of Repair Strategies**, categorizing how novices linguistically navigate technical breakdowns.

The identification of the **"Singin.html" Phenomenon**, a specific architectural misconception where users attempt to manage state via file creation.

The **"Aesthetic Compensation" Theory**, explaining why users pivot to visual polishing when functional logic fails.

2. Related Work

2.1 Large Language Models in Software Engineering

Recent advancements in LLMs, exemplified by tools like OpenAI's Codex and StarCoder, have revolutionized code generation (Chen et al., 2021). Studies by Vaithilingam et al. (2022) have shown that these tools significantly improve developer productivity by automating boilerplate code. However,

existing research predominantly focuses on **algorithmic correctness** in backend languages (e.g., Python, Java), where success is binarily defined by passing unit tests. Less attention has been paid to **Frontend Web Development**, where "correctness" is subjective, visual, and highly context-dependent. Furthermore, most studies recruit professional developers. Our work shifts the focus to **Novice End-User Programmers**, investigating how those without a mental model of "variables" or "functions" navigate the complexities of full-stack application construction.

2.2 The Challenge of End-User Prompting

As generative tools democratize access to coding, "Prompt Engineering" has emerged as a critical barrier. Zamfirescu-Pereira et al. (CHI '23) in their seminal work *"Why Johnny Can't Prompt"* highlighted that non-experts often struggle with **overgeneralization**—assuming that if the AI understands one concept (e.g., "blue button"), it understands the entire context. While their study focused on text-based chatbots, our research extends this to the **Visual-Functional domain**. In web development, users must describe spatial relationships ("move this to the right") and dynamic behaviors ("update the list after deleting"). These concepts suffer from **Spatial Deixis Failure**, where natural language lacks the precision to describe layout constraints (e.g., distinct from margin, padding, or float), leading to unique frustrations not seen in text generation tasks.

2.3 Mental Models of Web Architecture and State

Central to the "Refinement Crisis" is the divergence between the actual architecture of the Web and the user's mental model of it. Mental models are internal representations of how systems work (Gentner & Stevens, 1983). For novice programmers, these models are often built upon the **"Desktop Metaphor,"** where files are discrete, stateful entities that retain information when opened or moved. In the context of Web development, the architecture is inherently **Stateless** (Fielding, 2000), meaning data does not persist across page navigations without explicit state management (e.g., LocalStorage, Session, or Database). Prior research in end-user programming (Ko & Myers, 2004) identified the "selection barrier," where users struggle to identify the correct technical components to achieve a goal. Our study extends this by investigating how a faulty mental model—viewing the Web as a collection of persistent files rather than a dynamic state machine—leads to catastrophic failures when users attempt to refine multi-page applications like our CRUD note-taking task. By identifying the specific "Singin.html" misconception, we contribute to the understanding of how AI-assisted coding may inadvertently reinforce these faulty mental models by fulfilling syntactic requests while ignoring semantic gaps.

3. Methodology

3.1 Participant Demographics and Recruitment

We recruited 200 vocational college students from a vocational college in Central China from a non-Computer Science major course titled "Introduction to Digital Media."

Inclusion Criteria

Participants were screened via a pre-study survey to ensure they had **zero prior experience** in HTML, CSS, or JavaScript. This "Tabula Rasa" (Blank Slate) condition is crucial for our study, as it ensures that their prompting strategies reflect pure natural language intent rather than "pseudo-code" thinking.

Tool Environment

Participants were divided into two groups using **Tongyi Lingma** and **Trae**. Both tools are state-of-the-art, Chinese-native AI coding assistants integrated into VS Code, eliminating language barriers as a confounding variable.

3.2 The Task: Why CRUD?

While previous studies (Kim et al., 2022) focused on static styling tasks, we specifically chose a **CRUD Note-Taking Application** to force interaction with **State Management**.

The task required five specific milestones:

Creation

Input text and add it to a list (Tests DOM manipulation).

Persistence

Data must survive a page refresh (Tests localStorage understanding).

Read/Sort

Display notes, ideally sorted by time (Tests Array logic).

Update

Edit existing note content (The distinct "State Persistence" challenge).

Delete

Remove a note (Tests Event Listeners and Array mutation).

This incremental complexity allowed us to observe exactly *where* the user's mental model diverged from the AI's execution.

3.3 Data Analysis: Qualitative Coding

We collected a corpus of over **3,000 lines of interaction logs**. Data analysis followed a **Grounded Theory** approach(Charmaz, 2006) to identify emergent patterns in user repair strategies.

Open Coding

Two researchers independently tagged log entries with descriptive labels (e.g., "Complaining about style," "Asking for file creation").

Axial Coding

These labels were grouped into broader categories. For instance, commands like *"Make it red"* and *"Move to right"* were categorized under **"Perceptual Refinement."**

Selective Coding

We identified core themes—such as the **"Singin.html" phenomenon** and **"Aesthetic Compensation"**—by analyzing the temporal sequence of prompts (e.g., identifying that aesthetic requests frequently spike following a functional failure).

4. Results: The Anatomy of Breakdown

4.1 Quantitative Overview: Success vs. Struggle

Among the N = 200 participants, the majority demonstrated the capability to build functional web applications using AI tools, though success levels varied significantly.

Completion Rates: 65% (n = 130) of participants successfully implemented the full "CRUD + Search" functionality. Another **25% (n = 50)** completed the basic "MVP" features (Create/Read/Delete) but failed to implement advanced logic like Search or Update. The remaining **10% (n = 20)** abandoned the task after repeated failures (Global Reset).

Interaction Effort

Participants engaged in an average of **15 interaction turns** (SD = ± 4.2). This high number indicates that "one-shot" generation is a myth for complex apps.

The "Invisible State" Barrier: The most pervasive breakdown was the "Data Persistence" issue. A staggering **80% (n = 160)** of participants encountered data loss upon page refresh. Original logs show a repetitive pattern of complaints: *"After I add a note and refresh, it disappears"*, This reveals a widespread failure in the AI's default strategy, which prioritizes DOM manipulation over localStorage implementation.

The "Visual Fidelity" Gap

Aesthetic and layout issues affected **50%** of users. Common complaints involved "ugly buttons," "misalignment," and "height collapse," triggering extensive visual repair loops.

4.2 A Taxonomy of Repair Strategies

Through qualitative coding of the 3,000 + prompt lines, we identified four distinct patterns of repair strategies that novices employ when the AI output fails to meet expectations.

Type I: Perceptual Refinement (The Aesthetic Overload): Users lacking CSS vocabulary rely on cultural metaphors.

Evidence: Participant Li: *"The delete button... Change it to rainbow colors, Hello Kitty button shape."*

Analysis

This reveals a **Semantic Gap** where AI interprets "concise" as minimalist code, while users intend visual elegance.

Type II: Behavioral Correction (The State-Blindness Struggle): Users focus on the disconnect between data and view.

Evidence: Participant Gao: *"Clicking modify should not make it disappear... cover the old note after modification."*

Analysis

Users understand the **symptom** (data loss) but not the **mechanism** (LocalStorage), leading to repetitive storytelling prompts.

Type III: Structural Deixis (Layout Struggles): Users struggle with layout stability using spatial language.

Evidence: Participant Zhang): *"Center the pagination module... change width to 1200px."*

Analysis

The recurrence of specific pixel values suggests users pivot to **Hard-coding Constraints** when vague descriptions fail.

Type IV: Global Reset (The Abandonment):

Evidence: *"Rewrite everything"* or *"Forget the previous code"* appeared in 10% of sessions.

Analysis

Indicates a low tolerance for **Cognitive Debt**.

4.3 The "Singin.html" Misconception: Architectural Mismatch

A pervasive phenomenon observed in the second batch of logs (e.g., participants **P12, P35, P42**) was the explicit request to *"Create a file named singin.html for note details."* Interviews and prompt analysis reveal that singin is not a typo for "Sign In," but a novice shorthand for a **"Single Note View."**

The User's Model (File-Based)

Novices view a web app like a file system. They believe that clicking a note "opens" a specific file (singin.html), carrying the content with it naturally, much like opening a .docx file from a folder.

The Web's Reality (Stateless)

The Web is inherently stateless. When the browser navigates from index.html to singin.html, the JavaScript memory (RAM) containing the note array is wiped.

The AI's Failure

The AI faithfully executes the *syntax* (< a href="singin.html">) but fails to implement the invisible *glue code* (URL Parameters or localStorage) required to re-fetch the data.

The Outcome: Users reported verbatim: *"Clicking the note jumps to the new page, but the content is gone/empty."* This is not a bug in code syntax, but a failure in **State Persistence**, causing a "Silent Crash."

4.4 Granular Analysis of CRUD Friction

By decomposing the task into granular operations, we found that user struggles are not uniform.

4.4.1 Create: The "Enter Key" and Timestamp Anxiety:

The "WeChat" Habit

Many users (e.g., **Chang Xinyi**) requested *"Press Enter to send note."* The AI often added a keydown listener but failed to prevent the default behavior (newline), resulting in bugs where a blank note was added simultaneously.

Timestamp Formatting

Users like **Participant Wang** were obsessed with *"Add recording time."* However, simply appending new Date() resulted in ugly strings. Users spent 5–10 turns just refining this string, treating the AI as a string formatter.

4.4.2 Read: The Pagination Paradox:

The Paradox

Despite dealing with small datasets (< 10 notes), a disproportionate number of users (e.g., **Participant p5**) demanded **"Pagination"**.

The Logic Conflict

AI often implemented pagination using slice(), but when users simultaneously asked for *"Newest notes on top"* (Sorting), the AI used reverse(). These two logic blocks frequently conflicted, leading to users seeing the *oldest* notes on Page 1.

4.4.3 Update: The Navigation vs. Modal Conflict:

Navigation Model

As noted in 4.3, users often requested a jump to singin.html.

The Saving Loop

When users realized data was lost on jump, they tried to pivot to an **Inline Edit** model. However, prompts like *"Make the document modifiable"* led the AI to simply add contenteditable="true" without backend logic, resulting in "Phantom Edits" that vanished on refresh.

4.4.4 Delete: The "Spatial Deixis" of the Red Button:

Deixis Failure: The "Delete" function generated the most visual-specific prompts. Users treated the button as a physical object. **Participant Liu** instructed: *"Move delete button to the right... make it red."* The AI often struggled to interpret "Right" (Float vs Flexbox), causing layout shifts where adding a timestamp pushed the button off-screen.

4.5 The "Whack-a-Mole" Effect: Systemic Regression Cycles

A unique and critical finding in our study is the **"Whack-a-Mole" phenomenon**—a regression cycle where repairing one feature leads to the breaking of another.

4.5.1 Case Study A: The "Modify" Button Loop (Participant Zhang):

The Conflict

The AI successfully generated the logic (onclick="editNote()") initially. However, when the user prompted *"Put them close together"* (Visual Constraint), the AI rewrote the HTML structure, wrapping buttons in a new < div> but "forgetting" to re-attach the event listener.

The Outcome

The button looked perfect but became unclickable. The user assumed the feature was broken and deleted it entirely.

4.5.2 Case Study B: The "Delete" Button Layout Crash (Participant Wang):

Step 1 (Logic Success)

The user successfully implemented the delete function.

Step 2 (Visual Prompt): User prompted: *"Add a timestamp."*

Step 3 (Regression)

The AI inserted the timestamp string into the note container. Due to lack of CSS space management, the long string pushed the "Delete" button out of the visible area.

User Reaction: "My delete button is gone!" The user assumed the logic was deleted, whereas it was merely visually displaced.

4.6 The Emotional Trajectory: Aesthetic Compensation

The logs reveal a distinct emotional arc, culminating in a phenomenon we term **"Aesthetic Compensation."**

Phase 1: The "Magic" Honeymoon (0–10 Minutes):

Evidence:**Participant Xue** wrote: *"Very interesting! I learned new knowledge from the errors, happy!"*

Analysis

The "Gulf of Execution" feels non-existent. The AI's ability to produce *something* creates an illusion of mastery.

Phase 2: The Friction Plateau (10–30 Minutes):

Evidence: **Participant Pei** scolded: *"What you generated is wrong, regenerate it."* **Participant Jia** despaired: *"The website crashed halfway through."*

Analysis

This marks the onset of the **Refinement Crisis**. The "Singin.html" errors and CRUD frictions accumulate here.

Phase 3: Resignation and "Aesthetic Compensation" (30 + Minutes):

Evidence

Unable to fix complex logic (like "Save Edit"), users pivoted entirely to superficial beautification. **Participant P3** requested *"Change theme to pink,"* and **Participant P6**asked for *"Add cloud images"* specifically *after* failing logic tasks.

Theoretical Implication

Users unconsciously attempt to compensate for the lack of *functional utility* (broken logic) by maximizing *hedonic value* (visuals). By making the broken site look "professional" or "cute," they regain a sense of closure.

5. Discussion

5.1 Bridging the Gulf of Execution: From "What Code" to "What Happened"

Our results reveal a critical gap in current AI-assisted development: the asymmetry between **Static Code Generation** and **Dynamic State Management**. When participants encountered a logical bug (e.g., "the delete button doesn't work"), they were forced to debug via natural language guessing ("Is the code wrong?"). We argue that for novice users, **code is not the feedback; behavior is.** The AI should not just output a corrected code block; it must verify and visualize the *consequence* of that code. The current paradigm of "Chat-to-Code" is insufficient for dynamic applications; we need a paradigm shift to "Chat-to-Behavior."

5.2 Design Implication I: The Logic Inspector

Based on the "State-Blindness" strategy identified in Section 4.2, we propose that future AI coding tools should embrace **"Text-to-Behavior Visualization."** Specifically for logical operations (CRUD), the IDE should implement a **"Logic Inspector"** that reifies invisible data flows.

Scenario: When a user reports "Can't Delete," instead of just regenerating the JavaScript, the AI should present a **visual state diff**: *User Clicked Delete -> Array Updated □ -> View Updated □*.

Impact

By explicitly visualizing *where* the logic broke (Data vs. View), the AI shifts the user's role from a "blind guesser" describing symptoms to an "informed verifier" identifying root causes. This effectively bridges the Evaluation Gulf

5.3 Design Implication II: Logic Guardrails against "Whack-a-Mole"

To mitigate the regression cycles described in Section 4.5 (The "Whack-a-Mole" Effect), we suggest a **"Logic-Style Separation"** mechanism. Current LLMs often regenerate the entire file based on a new prompt, introducing the risk of overwriting existing logic.

The Problem

When a user prompts *"Make the button pink,"* the AI rewrites the HTML element. In doing so, it might accidentally drop the onclick="deleteNote()" attribute or change the class name that the JavaScript was listening to.

The Solution

Future IDEs should allow users to **"Pin" or "Lock" verified functional blocks**. When a user prompts for a visual change, the AI should be technically constrained to only regenerate the presentation layer (CSS) while preserving the locked logical integrity (JS/HTML structure). This feature would serve as a "Guardrail" for novice developers, giving them the confidence to iterate on aesthetics without fear of breaking functionality.

5.4 Theoretical Implication: The Trap of "High-Fidelity" Hallucination

The "Singin.html" phenomenon and the "Aesthetic Compensation" behavior point to a deeper issue: **High-Fidelity Hallucination.** Traditional prototyping tools (like Balsamiq) are intentionally low-fidelity to force focus on logic. AI tools, however, generate **High-Fidelity UI** (polished CSS, real buttons) from the very first prompt. This creates a dangerous illusion of completeness.

The user sees a "Sign In" button and assumes the "Sign In" logic exists.

The user sees a "Detail Page" (singin.html) and assumes data transfer logic exists. This visual realism masks the logical emptiness underneath. Future AI tools must consider **"Progressive Fidelity"**—perhaps rendering new features as "Wireframes" first to force the user to verify the logic before applying the "Pink Theme."

5.5 Limitations and Future Work

Our study is subject to several limitations that outline directions for future research.

Participant Demographics

Our participants were vocational college students (novices). Professional developers might employ different prompting strategies (e.g., pasting specific API documentation or error codes directly), which could lead to a different taxonomy of repair strategies.

Tool Specificity

We used *Tongyi Lingma* and *Trae*. While these represent state-of-the-art IDE agents, other models (like GPT-4o or Claude 3.5 Sonnet) might have lower hallucination rates. However, we argue that the fundamental "Logic-Visual Gap" is an epistemological problem regarding how users articulate intent, not just a model performance problem, and is likely to persist across platforms.

Short-term Study

The study focused on a single session. We do not know if users would learn to prompt better over time (Longitudinal effects). Future work should investigate how these strategies evolve as novices gain experience and whether the "Whack-a-Mole" effect diminishes with practice.

6. Conclusion

In this large-scale study of 200 novice developers, we demonstrated that while AI can easily generate the syntax of a web application, it struggles to help users manage the semantics of application state. We identified four key repair strategies—Perceptual, Behavioral, Structural, and Reset—and characterized the "Whack-a-Mole" phenomenon as a critical friction point in multi-modal generation. The "Refinement Crisis" we observed suggests that the future of AI programming tools must evolve from "Code Generators" to "Logic Visualizers." Only by reifying the invisible logic and providing guardrails against regression can we truly empower end-users to see not just the code they wrote, but the reality they created.

Declarations

Ethics declaration: This study involved human participants and was approved by the Institutional Research Committee of Anyang Preschool Teachers College.

Ethics approval:

This study involved human participants (200 students) and was conducted in accordance with the ethical standards of the institutional research committee at Anyang Preschool Teachers College.

Consent to participate:

Informed consent was obtained from all individual participants included in the study.

Consent to publish:

The authors affirm that human research participants provided informed consent for the publication of the research findings and any accompanying images in an anonymized format.

Competing interests:

The authors have no relevant financial or non-financial interests to disclose.

Funding:

The authors received no specific funding for this research. The study was conducted as part of the authors' educational and research activities at Anyang Preschool Teachers College.

Author Contribution

A wrote the main manuscript text

References

1. Zamfirescu-Pereira JD, Wong RY, Hartmann B, Yang Q. (2023). Why Johnny Can’t Prompt: How Non-AI Experts Try (and Fail) to Design LLM Prompts. Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems (CHI '23).
2. Kim T, Bragg J, Chilton LB. (2022). Stylette: Styling the Web with Natural Language. Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems (CHI '22).
3. Ko AJ, Myers BA. (2004). Six Learning Barriers in End-User Programming Systems. Proceedings of the 2004 IEEE Symposium on Visual Languages - Human Centric Computing (VL/HCC '04).
4. Vaithilingam P, Zhang T, Glassman EL. (2022). Expectation vs. Experience: Evaluating the Usability of Code Generation Tools Powered by Large Language Models. Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems (CHI '22).
5. Chen M, Tworek J, Jun H et al. (2021). Evaluating Large Language Models Trained on Code. arXiv preprint arXiv:2107.03374.
6. Norman DA. Cognitive Engineering. User Centered System Design. New Perspectives on Human-Computer Interaction; 1986.
7. Barke L, James MB, Polikarpova N. (2023). Grounding Copilots in Human Interaction. Proceedings of the ACM on Programming Languages (OOPSLA '23).
8. Charmaz K. Constructing Grounded Theory: A Practical Guide through Qualitative Analysis. Sage; 2006.
9. Gentner D, Stevens AL. (1983). Mental Models. Psychology.
10. Fielding RT. (2000). Architectural Styles and the Design of Network-based Software Architectures. Ph.D. Dissertation, University of California, Irvine.

Figures

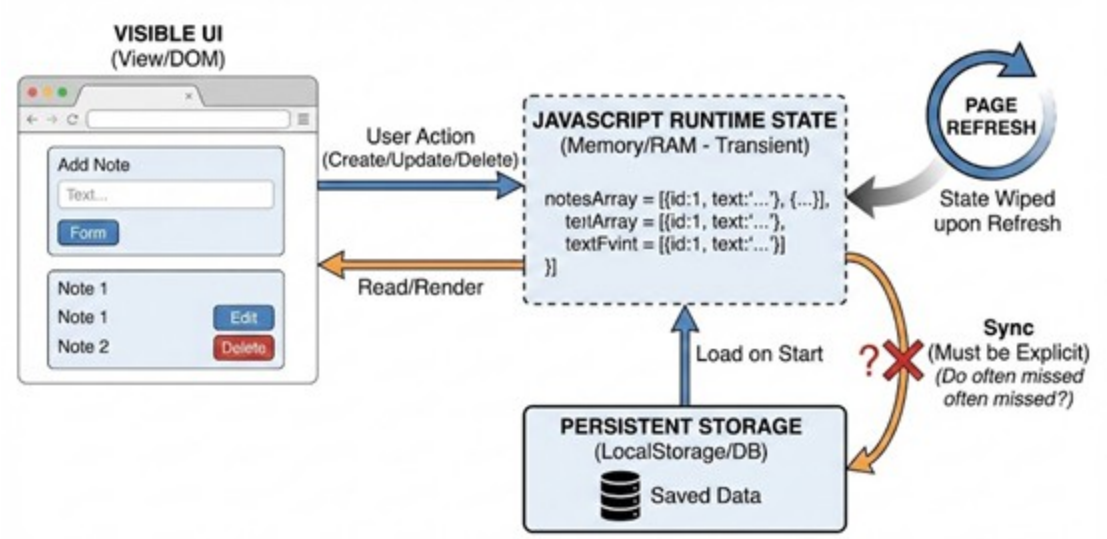


Figure 1: Web Application CRUD State Architecture & The "Invisible State" Gap

Figure 1

See image above for figure legend.

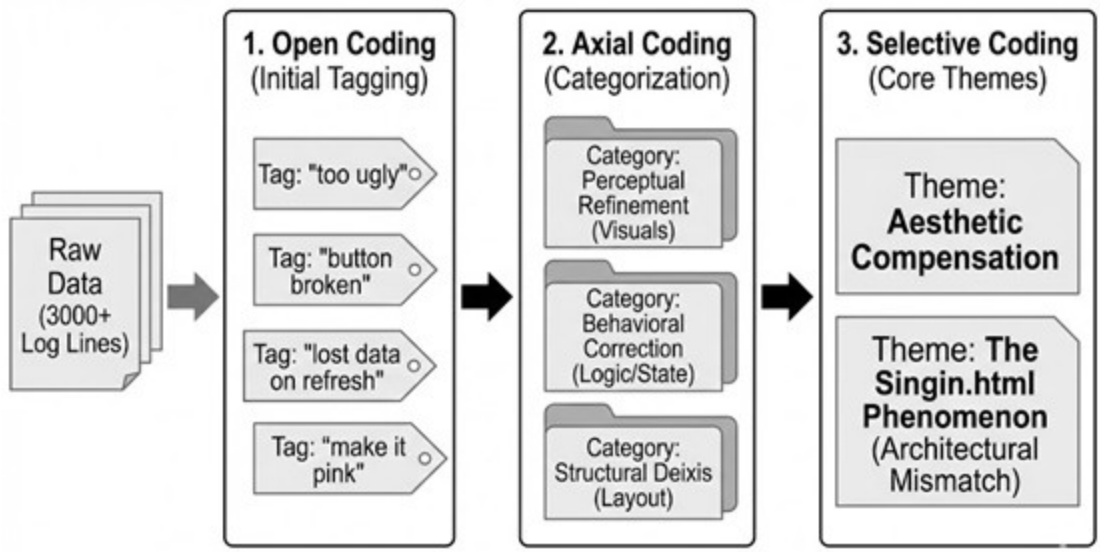


Figure 2: Grounded Theory Qualitative Data Analysis Process

Figure 2

See image above for figure legend.

Supplementary Files

This is a list of supplementary files associated with this preprint. Click to download.

- Appendices.pdf