THE VIBE DEPARTMENT / FIELD NOTES 002

GOT VIBES?

THE VIBE-TO-PROOF HANDBOOK

THE IDEA

You can prototype through language. You still need judgement, evidence, maintenance and respect for the people who will use what you make.

A practical, original companion for experimental builders, creative generalists and people who supervise AI-generated code.

A SAHILTAYAL.COM PRODUCTION | NOVEMBER 2026 | ORIGINAL MATERIAL: CC BY 4.0

01 / THE PREMISE

The magic trick has a maintenance bill.

Natural-language development can lower the friction of starting. The responsibility for the result does not vanish when an AI tool produces convincing code.

The distinction worth keeping

- **Idea:** an imagined benefit for a particular person.
- **Prototype:** something that demonstrates a possible interaction.
- **Product:** a system people can rely on for a defined purpose.
- **Production:** the real operational environment, with ongoing obligations.

These are not the same stages of proof. A working demonstration tells you that a path works under observed conditions; it does not prove that the application is secure, accessible, truthful or suitable for higher-stakes use.

The three human responsibilities

- **Direction:** state whose problem matters and what is excluded.
- **Verification:** inspect the output, test it and distinguish observed behaviour from assumptions.
- **Stewardship:** assign maintenance, data handling and safe rollback to a real person.

> **NOTE**
>
> The phrase “vibe coding” is informal, not a technical certification. The process in this handbook is an educational practice model; it does not replace specialist review.

02 / SCOPE

First decide what should not exist.

The easiest way to create an unmaintainable app is to ask for everything at once.

The one-action rule

Try completing this sentence: “A visitor comes here to _____, and leaves able to _____.” If you cannot complete it without adding five features, the first release is too broad.

A practical risk spectrum

- **Static exhibit:** limited data and consequence. Prioritise semantics, mobile performance, links and content accuracy.
- **Business forms:** add consent, storage, anti-abuse protection, delivery reliability and clear ownership.
- **Account or transaction system:** require robust authentication, authorization, recovery, vendor review and qualified security scrutiny.
- **High-consequence system:** add domain experts, governance, independent testing and documented limits. An AI prototype is not adequate assurance.

Write three non-goals

Examples: no accounts, no external API calls, no personal information. Non-goals are a design tool: they prevent new dependencies from arriving disguised as nice-to-have features.

WORKSHEET

User: _____ Core action: _____ Observable result: _____

Three non-goals: 1. _____ 2. _____ 3. _____

03 / SPECIFICATION

Make the idea testable.

A usable brief is not a long mood board. It explains behaviours and boundaries in a way that another person can test.

The minimum specification

- Audience and situation: who is using the thing and why?
- Primary flow: start, action, result, recovery.
- States: loading, empty input, errors, offline and unexpected conditions.
- Inputs and outputs: what arrives, what is stored and what is displayed?
- Accessibility: labels, keyboard order, focus, contrast and motion preferences.
- Acceptance checks: statements a human can confirm or reject.

Example: a museum for abandoned ideas

A visitor types a fictional idea, selects one of three ceremonies, and sees a thoughtful farewell. The first version stores nothing and uses no server calls. The farewell text must not interpret input as HTML. A keyboard-only visitor can complete the flow. On a narrow screen, no horizontal scrolling is required.

SPECIFICATION TEST

Replace “make it delightful” with “a first-time visitor can complete the core flow on a 360px-wide screen using only the keyboard.” The second statement is testable.

04 / PROMPT ANATOMY

Ask for boundaries, not miracles.

Good prompts are structured requests. They cannot remove the need for code review or independent evidence.

Six useful components

- **Role:** “Act as a critical implementation reviewer.”
- **Purpose:** the user, action and outcome.
- **Constraints:** runtime, dependencies, accessibility, privacy and licensing.
- **Acceptance:** observed, testable conditions.
- **Exceptions:** request handling of error, missing and adversarial cases.
- **Output:** files, run commands, tests and known limitations.

Starter prompt

COPY / ADAPT

“Build one small vertical slice of [USER ACTION] for [AUDIENCE]. First state assumptions, data flows and non-goals. Prefer semantic HTML and accessible native controls. Avoid new dependencies and client-side secrets. Show a file plan, tests, local run steps and unresolved risks. Do not claim production success without live evidence.”

Ask the model to distinguish: implemented, tested, inspected, assumed and unknown. These words should not be treated as synonyms.

05 / BUILD QUALITY

A thin slice beats a tall promise.

An end-to-end slice contains one complete path through the system, not a disconnected set of fancy screens.

Prefer boring clarity in the foundation

- Readable file names and a clear run command.
- Dependencies you understand, can audit and can update.
- Safe text rendering: user text as text rather than arbitrary HTML.
- A sensible fallback when data is empty or an API fails.
- Visible focus, labelled controls and mobile-first layouts.

The incremental review loop

Implement one flow. Run it. Read the code. Break it intentionally. Save the passing revision. Only then add the next feature. This reduces the surface area of uncertainty and makes rollback more practical.

PROOF NOT PERFORMANCE

“The code compiles” is one fact. “A person completed the task on a mobile device” is another. “The app is safe for production” requires a much broader body of evidence.

06 / SECURITY

Assume an input will surprise you.

A text box, URL parameter, upload, API response or database field can carry unexpected content. Generated code needs the same careful boundaries as handwritten code.

Common failure categories

- **Injection:** rendering unsafe user content as markup or sending untrusted input into commands/queries.
- **Broken authorization:** trusting the interface to enforce who may see or change data.
- **Secrets exposure:** shipping credentials inside browser-delivered code or repositories.
- **Unreviewed dependencies:** introducing components with unknown updates, licences or vulnerabilities.

A safer first pass

Minimise what you collect. Use established authentication and payment providers where appropriate. Keep secrets on the server in protected configuration. Test access checks server-side. Treat the OWASP Top 10 as an awareness starting point, not a complete audit.

STOP CONDITION

If the project can expose personal information, transfer money, affect safety or create material harm, do not infer safety from AI-generated explanations. Seek qualified review.

07 / PRIVACY AND TRUTH

Data collection changes the obligations.

A prototype that records nothing is not equivalent to a service that stores names, emails, behaviour or financial details.

The minimum data inventory

- What data enters the system, and through which form, API or third party?
- Why is each field necessary? Who can access it?
- Where is it stored, for how long, and how is it deleted?
- What does the user know and consent to? Which laws may apply?

The fact-checking boundary

The fluent voice of an AI assistant can make unsupported claims sound finished. For statistics, quotes and studies, locate the source. Inspect method, scope, publication date and uncertainty. Remove facts you cannot verify instead of inventing a bibliography.

BETTER COPY

Instead of “AI-generated applications are always faster and cheaper,” say “AI tools may reduce the time required for some prototypes; the benefits depend on task, experience, integration and review costs.”

A beautiful control must also be usable.

Accessibility is an ongoing practice: a scanner can help, but keyboard use, content clarity and real user needs require human attention.

Try these tests yourself

- Put the mouse away. Can you reach, activate and leave every interactive element?
- Open a modal or menu. Where does focus go? Can Escape dismiss it?
- Increase zoom to 200%. Does essential content disappear?
- Try reduced motion. Is nonessential movement suppressed?
- Check labels, instructions, errors and contrast in context.

Build from semantic foundations

Use native links for navigation, buttons for actions, labels for form fields and appropriate headings. These defaults reduce avoidable complexity. Consult WCAG 2.2 for testable success criteria and techniques; meeting every user need takes more than a checklist.

DESIGN CONSTRAINT

“Include a keyboard-operable accessible alternative” is a stronger design requirement than “make it look futuristic.”

09 / TESTING

Try to disprove the happy path.

Test the error path, not only the demonstration that already works.

A compact test matrix

- **Viewports:** small phone, larger phone and desktop.
- **Input:** empty, lengthy, unusual characters and adversarial-looking values.
- **Interaction:** pointer, keyboard and zoom.
- **Conditions:** slow network, missing resources, errors and disabled third parties.
- **Claims:** external links, references, downloads and ownership statements.
- **Deployment:** exact route, adjacent paths, redirects and the public URL.

Log a failure as reproducible evidence

Record the environment, steps, expected result, observed result, impact and screenshot/log. Re-run the same steps after the fix. Use automated checks for repeatability, but include real browser observation.

TEST RECORD

Case: _____ Environment: _____ Steps: _____
Expected: _____ Observed: _____ Fix / retest: _____

10 / RELEASE

A deploy is not the same as a launch.

Publish one known revision to a known destination with a way back.

The release sequence

- Identify the domain, exact path and resource currently owning that route.
- Confirm that the change will not replace an unrelated production resource.
- Build an isolated preview and inspect it in a browser.
- Record the exact source revision or build hash.
- Obtain the responsible owner’s approval and use least-privilege credentials.
- Publish once; inspect status and content at the final live URL.
- Exercise an important user flow and verify neighbouring routes.
- Retain and test the rollback path.

When automation is blocked

A carefully documented manual release can still be responsible. Do not equate “manual” with “skip verification.” The same route ownership, approval, preflight, production smoke test and rollback rules apply.

RELEASE RECORD

Owner: _____ Exact route: _____ Source revision: _____
Preview checked: _____ Production observed: _____ Rollback: _____

Your page is alive after it ships.

Maintenance includes code, dependencies, third-party changes, content, hosting, feedback and sometimes a decision to shut something down.

A minimal care ledger

- **Owner:** a real person who receives failure reports.
- **Surface:** routes, forms, domains, APIs, data stores and external links.
- **Cadence:** periodic quick checks appropriate to importance and risk.
- **Incident response:** how to pause, revert, notify or correct.
- **Retirement:** how to remove the service and its data safely.

The overlooked cost

Short prototype time can coexist with long maintenance effort. Every added dependency, feature and route creates a future obligation. A compelling idea earns its place by being useful enough to justify care.

MAINTENANCE ENTRY

Next review date: _____ Who checks: _____ Highest-risk failure: _____
Recovery method: _____ Known issues: _____

Say exactly what you know.

Confidence is more useful when it is attached to an observable source.

Five different sentences

- **Designed:** a planned feature or layout exists in a specification.
- **Implemented:** code for that feature exists.
- **Locally tested:** a specified check passed in a specified environment.
- **Production verified:** the intended result was observed on the real public route.
- **Independently reviewed:** a qualified person outside the implementation process assessed relevant risks.

One level does not magically imply the next. A locally running prototype is not a live service; a live service is not automatically secure; and a single green check is not a timeless guarantee.

Documentation that earns trust

Keep short records: what changed, why, who approved, what was tested, what failed, which revision is live and what remains unresolved. The goal is traceability rather than ceremonial paperwork.

> **A USEFUL CLAIM**
>
> “I verified the public route, its main interaction and the downloadable guide on 18 November 2026” is more meaningful than “everything is perfect.”

Leave with a better question.

Use the browser page for interactive versions and downloadable Markdown. These small patterns are enough to start an accountable build.

The pre-build question

Who is the visitor? What is the one task? What data will be touched? What could go wrong? What would show the task works? What is deliberately excluded?

The reviewer question

What assumptions have not been tested? Which failure states have not been seen? Which claims lack sources? What could a careless or malicious user do? What does the accessibility path look like?

The release question

What exactly is going live, at what URL, under which owner, after which checks, with which permissions, and how will we reverse it if production disagrees?

THE POCKET RULE

FRAME. SPECIFY. BUILD. BREAK. RELEASE. MAINTAIN.
The order can loop. Accountability cannot be delegated to a text generator.

14 / REFERENCE SHELF

Read the originals.

This handbook is an original educational synthesis, not a substitute for authoritative engineering or domain guidance.

W3C / Web Content Accessibility Guidelines 2.2
w3.org/TR/WCAG22/

OWASP / Top 10 Web Application Security Risks (2025)
owasp.org/projects/top-ten

MDN / Cross-site scripting (XSS)
developer.mozilla.org/en-US/docs/Web/Security/Attacks/XSS

web.dev / Web Vitals
web.dev/articles/vitals

NIST / AI Risk Management Framework
nist.gov/itl/ai-risk-management-framework

GitHub Docs / GitHub flow
docs.github.com/en/get-started/using-github/github-flow

FINAL NOTE

Make things. Invite curiosity. Keep the testing receipts. A machine can help make an interface; a person remains accountable for what happens when others rely on it.

SAHILTAYAL.COM / RESEARCH MEETS ART / ORIGINAL MATERIAL CC BY 4.0