AI AGENT TO SHIPPING

SOFTWARE ENGINEERING FOR SHIPPING VIBE-CODED PRODUCTS

A Solo Builder's System for Planning, Architecture, TDD, Review, Deployment, and Operations with AI Agents

CHANGE CONTRACT · TEST EVIDENCE · FAIL-CLOSED · RELEASE GATES

iisacc

First English Edition · 1.0.0

Contents

Opening

The Promise of This Book

This book does not sell a "magic prompt that builds an app from one sentence." It does not recommend deploying code that AI produced quickly but that its operator does not understand. Instead, it explains how to use AI as a change proposer, researcher, implementer, and verification assistant while a human retains authority over product direction and quality.

The scope does not end when an idea appears on a screen. It continues through writing requirements as changeable contracts, turning the repository into the agent's working memory, evaluating external dependencies, fixing small changes with tests, inspecting real builds and user surfaces, closing payment, security, and deployment failures safely, and reducing operating cost after launch.

Who This Book Is For

- People who can prototype with AI coding tools but see their existing codebase deteriorate with every change
- Solo business owners who must plan, build, design, deploy, and sell a product themselves
- Developers who want to delegate work safely to AI agents inside an existing repository
- Teams that need a product-level definition of done covering tests, deployment, security, and payments
- Builders who want an operating system that survives changes in models and tools

You can understand the central principles without deep programming knowledge. This book does not promise that "anyone can succeed without learning development." In the age of vibe coding, development skill includes the ability to investigate what you do not know, divide risky changes into small units, and refuse to ship unverified results.

Notes on Examples, Counts, and Translation

The examples come from the July 31, 2026 source snapshots of Vincent, iiPaintEngine, LVRS, WeUs/AISNS, iisacc.com, and Time Scopes, all developed by iisacc. Each repository uses a different language and testing system, so a test declaration count in one project is not treated as equivalent to a test file count in another. Only structures and decisions that can be disclosed are used. API keys, authentication material, private prompts, and customer data are excluded.

This English edition was produced through an AI-generated translation workflow and reviewed and edited by iisacc. The Korean first edition remains the source work. Technical names, commands, product facts, and rights statements were checked against that source rather than inferred from the translation.

Copyright and Permitted Use

Part I. Turning Vibes into Engineering

1. Writing Code Quickly Is Not the Same as Shipping a Product Quickly

The first experience of vibe coding can feel like the disappearance of a bottleneck. Tasks that once required hours—creating files, reading framework documentation, looking up APIs, and writing repetitive code—can now be described in natural language and turned into dozens of files in minutes. A screen appears. The buttons move. At that moment it is easy to confuse code-generation speed with product-development speed.

Product speed is not typing speed. It is the total time required to choose the right problem, understand the scope of a change, preserve existing behavior, deliver the result somewhere a user can actually reach, and recover when something goes wrong. Code created in ten minutes does not make a fast product if it breaks only in production and takes two days to diagnose. An implementation that takes two hours can still be fast when its test and deployment contracts are clear enough to ship it once.

Remember the difference with this expression:

$$\text{Product delivery velocity} = \text{generation speed} \times \text{probability of the right direction} \times \text{verification confidence} \times \text{deployment success rate}$$

Each factor is between zero and one. Even if generation becomes ten times faster, actual delivery will not rise as expected when the other three factors fall by half. AI makes plausible code in volume, so it amplifies the cost of a wrong direction as well as the speed of a correct one.

It Works Has Several Layers

When an AI says that a task is complete, decompose what was actually completed.

- Source aligned: the code structurally matches the requirement.
- Static verification: type checking, linting, and compilation pass.
- Dynamic verification: the relevant tests actually pass.
- Packaging verified: the bundle, installer, or web build that the user receives is produced.
- Deployment verified: the production environment serves the new artifact.

- User surface verified: the result appears through the real browser, app, payment, or download path.
- Operations verified: logging, retry, cancellation, and recovery behave as expected.

Once these layers are separated, “the button works locally” and “a customer can pay and receive the file safely” are clearly different states. Vibe coding becomes dangerous when an AI can cover the gaps between those states with smooth prose. Engineering names each gap as a state and requires evidence for every state.

How Prototype Debt Appears

Prototypes are not bad. The problem begins when a prototype is promoted to an operating product without changing its contract. Temporary data, hard-coded URLs, secrets in the client, an installation process that succeeds only once, and manually tuned UI remain because they “work for now.” AI is highly effective at satisfying the visible success condition. If you do not explain the future reasons for change, it will often bind several responsibilities together.

A request for a payment button, for example, can touch the UI, price display, provider configuration, checkout session, webhook verification, purchase entitlement, download, refund, and cancellation. If the screen is the only acceptance criterion, an AI may treat the client-side checkout.completed event as payment success. That event is a navigation signal, not the authority for payment. The server must not issue an entitlement until it verifies the transaction through the provider API or a signed webhook.

The answer is not a longer prompt. The answer is to identify who owns each product fact and to define the default state on failure.

```
Completion contract
1. Use the browser completion event only for navigation.
2. Create purchase access only after the server verifies the provider transaction.
3. Fail closed unless price, currency, product ID, and transaction state all match.
4. Repeated delivery of the same webhook must create only one entitlement.
5. Prove tests, the build, and the real callback path independently.
```

These five lines are shorter and stronger than “implement payments.” They establish invariants without freezing a particular implementation.

The Goal Is Less Rework, Not Slower Work

Writing a contract before a change, adding a small test, and checking a build can look slow. In reality, they move the hidden cost of post-generation rework to the beginning. A good vibe-coding workflow does not maximize the amount of code an AI writes. It shortens this loop:

1. Choose one observable problem.
2. Record the current and desired behavior as evidence.
3. Define the smallest change boundary.
4. Create or reproduce a failing verification.
5. Implement the change.
6. Pass narrow verification first, then broad verification.
7. Inspect the real user surface.
8. Update documentation and operating contracts.

When this loop is small, a misunderstanding by the AI produces limited damage. Bad assumptions surface quickly, and the human can review a bounded difference instead of rereading the whole codebase. Generation stays fast while promotion of a change remains deliberate.

Exercise 1: Separate the Layers of Completion

Choose one feature you are currently building. Avoid broad labels such as "finish signup," "deploy the app," or "integrate payments." Record its state with this form:

```
Feature: _____
Source-alignment evidence: _____
Static-verification evidence: _____
Dynamic-verification evidence: _____
Packaging evidence: _____
Deployment evidence: _____
User-surface evidence: _____
Operations/recovery evidence: _____
Not yet verified: _____
```

An empty field is not a failure. Hiding that field behind the single word "done" is the failure.

2. AI Is Neither a Co-Founder nor an Autopilot

Using conversational language with an AI agent is convenient. "Investigate this," "fix it," and "take it all the way to launch" feel like collaboration. Product responsibility, however, cannot be delegated with the work. AI is a powerful operator, but it does not own the business objective, legal responsibility, customer promise, or decision to accept risk.

The most practical role definition is this:

> AI gathers evidence, proposes changes, and acts within an authorized boundary. A human owns the objective, priorities, authority boundary, and promotion of claims into facts.

This definition does not reduce AI to passive autocomplete. It permits wide delegation across research, implementation, testing, documentation, and deployment verification. The deciding evidence simply comes from the external world, not from the AI's sentence.

Four Modes of Work

AI work becomes safer when divided into four modes.

1. Exploration Mode

Read the repository structure, relevant files, official external documentation, and current deployment state. Change no files. Produce hypotheses and supporting evidence. The less clear it is where to make a change, the more important exploration becomes.

2. Diagnosis Mode

Separate symptoms from causes. Compare reproduction conditions, logs, recent changes, and data boundaries. A diagnosis request does not grant modification authority by default. Changing code while looking for the cause moves the evidence and can make the investigation slower.

3. Change Mode

Modify code, tests, and documentation together inside the authorized scope. The completion contract matters more than a fixed list of target files. An agent may discover a relevant file that no one anticipated.

4. Operations Mode

Change external state through builds, deployments, product registration, monitoring, or customer delivery. Before acting, ask whether the action is reversible, whether repetition is safe, and whether failure can damage existing customers.

One request can require all four modes. "Fix the bug and deploy it" includes exploration, diagnosis, change, and operations. Require the agent to retain evidence from each stage.

Authority and Verification Are Not Opposites

Giving an AI autonomy is not the same as skipping verification. Autonomy is the ability to choose the next safe action inside the objective without waiting for approval before every click. Verification is the ability to prove that the action succeeded in the external world.

A useful delegation contract looks like this:

```
Objective: Keep the notification badge consistent with the read state.
Authority: Modify relevant source, tests, and documentation, and run local verification.
Constraints: Do not change secrets or production data. Do not refactor unrelated features.
Done: Prove the reproduction test, full typecheck/lint/test/build, and the real screen state.
Report: Separate changed facts, evidence, and external states that still require outside authority.
```

Inside that contract, an AI may locate files, modify them, and repeat tests. Results outside its control—such as App Store approval, a real card payment, or receipt by another person's mailbox—must remain "unverified."

Three Illusions AI Commonly Creates

The first is the narrative illusion. A long explanation and a confident summary can look like evidence. Counter it by requiring command output, file locations, test results, and the state of the real URL.

The second is the near-success illusion. The system infers that production must work because the configuration looks correct. A DNS record existing does not prove that a custom domain is active. A successful build, a response from a platform default domain, a connected custom domain, and transfer of DNS authority are separate states.

The third is the substitute-success illusion. When the original path is blocked, the system points to the success of a similar path. It may replace the required authenticated Chrome session with an unauthenticated embedded browser, or