The 4-Week AI MVP Playbook

From idea to deployed, production-ready AI product in 28 days. The complete system — including the parts most build guides skip entirely.

Most AI products don't fail because the idea was bad. They fail because the team spent four months building the wrong version of a good idea.

This playbook exists to prevent that. It is built on a single conviction: the fastest path to knowing whether your AI product works is to have real users using a real version of it — not a Figma prototype, not a ChatGPT wrapper in a staging environment, not another planning session.

Twenty-eight days. Here is exactly how.

Who This Is For

This playbook is written for founders, operators, and product leaders who have a validated AI product idea and need a structured system to take it from concept to deployed reality in one month.

It is not for teams who are still in the "exploring whether AI is right for us" phase — the $200K Mistake Report handles that question. It is for teams who have answered yes and need to move.

It assumes you have: a clear problem statement, a target user whose pain you understand firsthand, a rough technical direction, and at least one person who can write and ship code. It does not require a large team. The companies who have used this system successfully range from solo technical founders to five-person product teams.

The Core Principle: The Ratchet Rule

Before the playbook begins, there is one rule that governs everything inside it.

Every day, something must be more real than it was the day before.

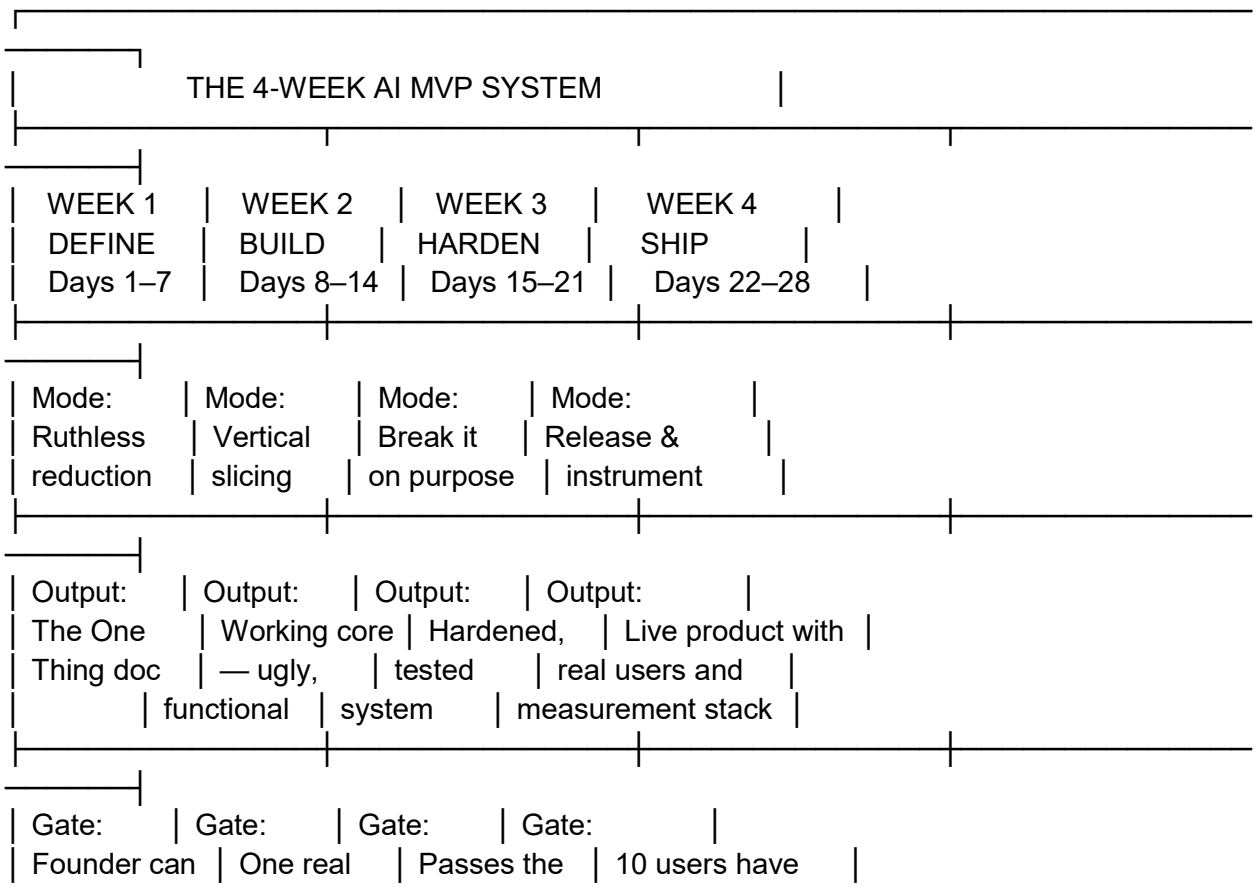
Not more planned. Not more designed. Not more discussed. More *real* — meaning closer to a state where a user can interact with it and produce a signal about whether it works.

This sounds obvious. It is almost universally violated. Teams spend day three of week one rewriting the project brief. They spend day four in a stack debate. They spend day five designing the database schema for features they haven't built yet.

The Ratchet Rule prevents this. Every morning, the first question is: what was the most real thing we had yesterday, and what will make it more real today? The answer to that question is the day's work. Everything else is noise.

The Framework: 4 Weeks, 4 Modes

Each week operates in a distinct mode with a distinct output. The mode change at the end of each week is not a calendar convention — it is a structural gate. You do not enter Week 2 until Week 1's output exists.



demo the	user has	Break Test	used it. You have
logic in	used it &	(defined	their feedback.
plain words	given signal	below)	

Week 1: Define

Days 1–7 · Mode: Ruthless Reduction

The job of Week 1 is not to plan the product. It is to destroy every version of the product that isn't the minimum viable one.

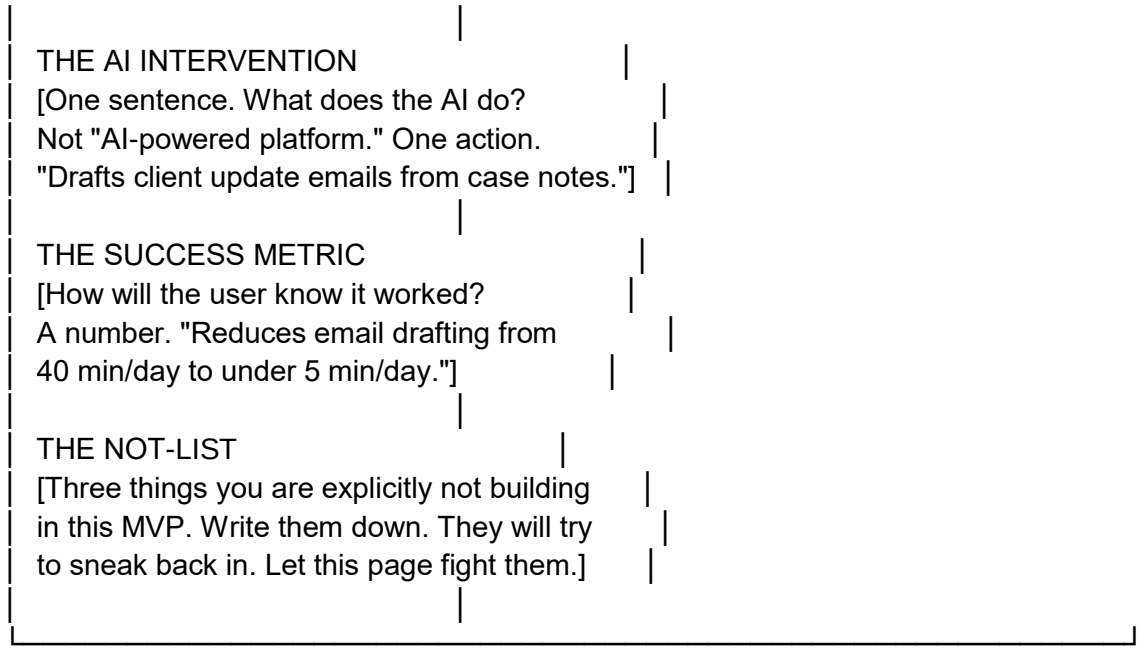
Most teams arrive at Week 1 with a product idea that is secretly three products. There is the core use case, the adjacent feature that would make it more compelling, and the future roadmap item that keeps sneaking into the conversation because it feels important. Week 1's job is to identify the core use case, kill everything else, and write it down so clearly that anyone who reads it can understand what you're building and what you're deliberately not building.

This is psychologically the hardest week. It requires saying out loud: we are not building that yet. For founders, this feels like giving up. It is the opposite.

The One Thing Document

By end of Day 3, you must have a One Thing Document. It is one page. It answers five questions only:

THE ONE THING DOCUMENT
THE USER
[One sentence. Specific person. Not "businesses." "A solo employment lawyer who manages 200+ client files without a paralegal."]
THE PAIN
[What are they doing manually right now that is costing them time, money, or accuracy? Be specific. Use numbers if possible.]

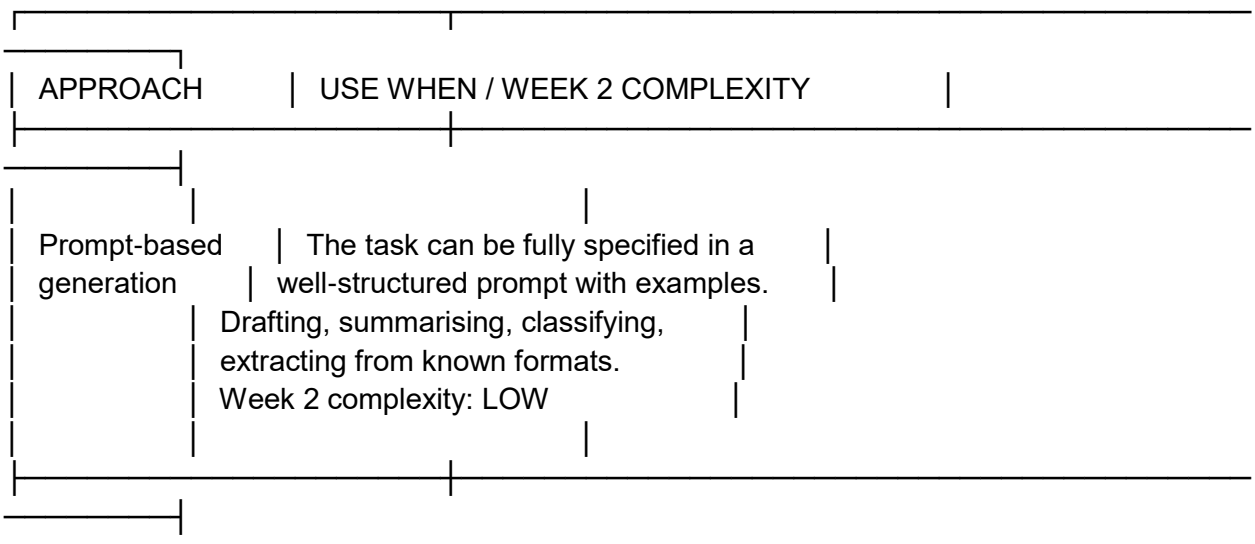


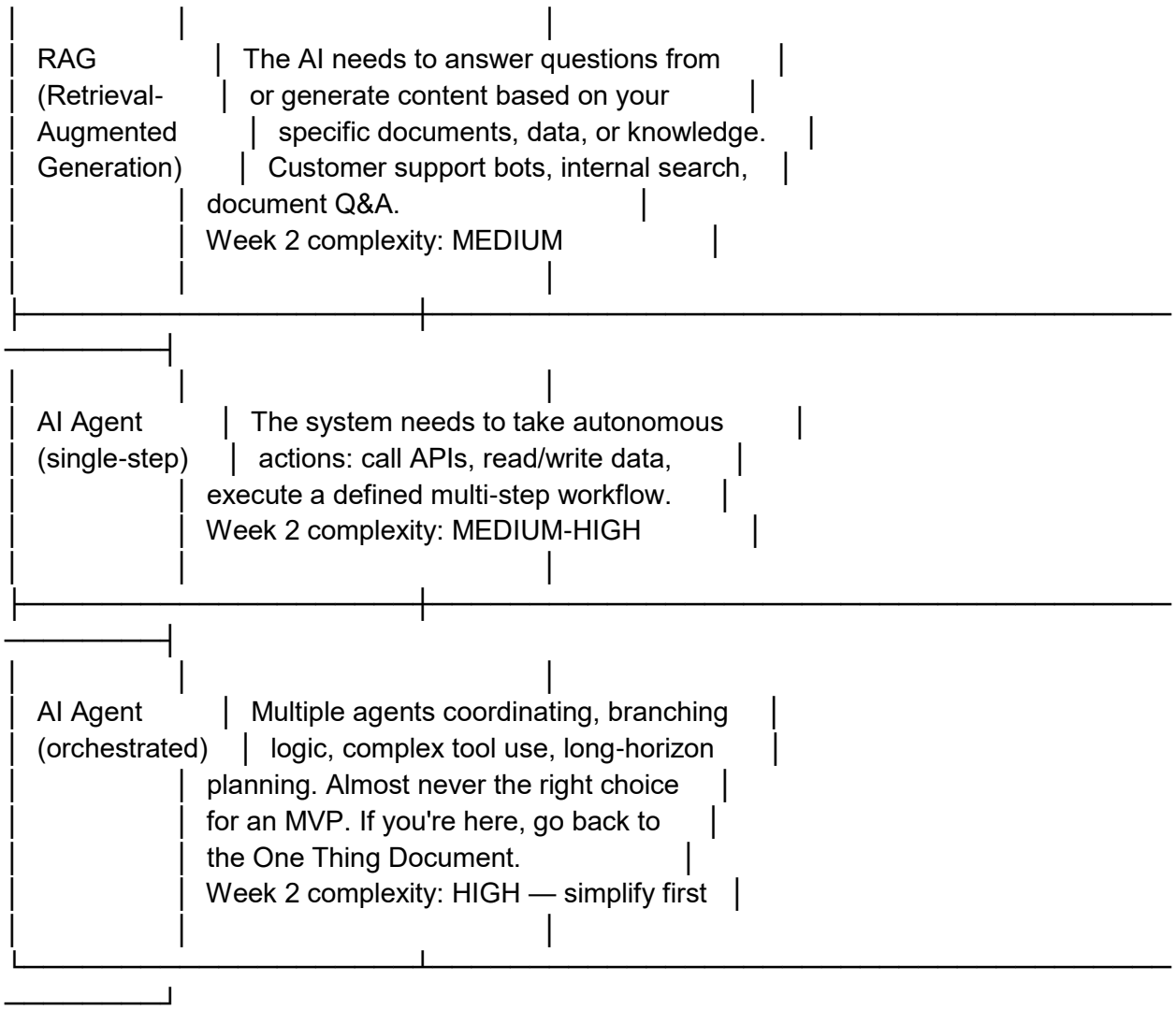
This document is not a PRD. It does not have user stories. It does not have a roadmap. It is a promise to yourself about what the next 21 days are for.

The Technical Direction Decision

Days 4 and 5 are for making one technical decision that will govern everything in Week 2: which AI approach does the core intervention require?

There are four categories, and they have meaningfully different implementation timelines:





The most common mistake at this stage: founders assume their idea requires an orchestrated multi-agent system because it sounds more powerful. In almost every case, a well-designed prompt-based or single-step agent approach delivers the same user value in a fraction of the time. Choose the simplest approach that delivers the core value. Complexity is not a feature.

Week 1 Days 6–7: The Paper Walkthrough

Before a single line of production code is written, you walk the entire user journey on paper. Not in Figma. On paper.

Every screen. Every AI call. Every input the user provides. Every output they receive. Every error state. Every empty state.

This sounds like unnecessary process. It saves, on average, 3–5 days of rework in Week 2 by surfacing the questions you didn't know you had. Where exactly does the AI call happen in the user flow? What does the user see while they wait for the response? What happens when the AI produces something unexpectedly long? What happens when it produces something wrong?

Week 1 Gate Check: Can the founder explain, in plain language with no technical terms, exactly what happens when a user sits down and uses the product for the first time? If yes: enter Week 2. If no: the One Thing Document needs more work.

Week 2: Build

Days 8–14 · Mode: Vertical Slicing

Week 2 has one goal: get one complete user journey working, end to end, in a real environment. Not a beautiful one. Not a complete one. One journey, working, real.

This is called vertical slicing — cutting through all the layers of the application (frontend, backend, AI layer, data) for a single use case rather than building each layer horizontally across all features.

The alternative — horizontal building — is what most teams instinctively do and what consistently produces Week 3 crises. They spend Week 2 building a complete authentication system, a complete database schema, a complete API layer. At the end of the week, they have excellent infrastructure and zero product. They have no signal from a real user because no real user can do anything yet.

The Vertical Slice Priority

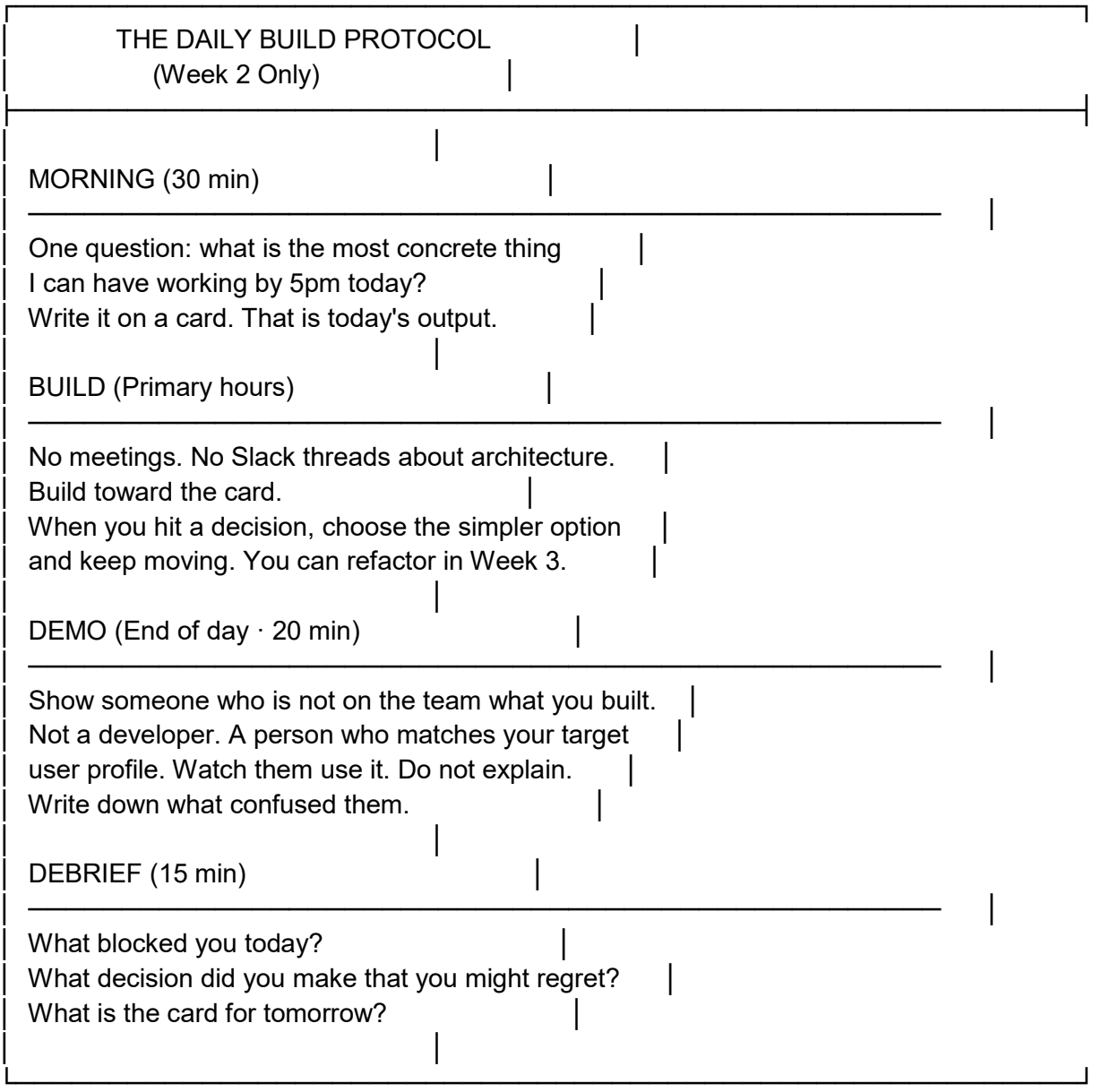
Not all user journeys are equal. The vertical slice you build in Week 2 should be the one that:

1. Delivers the core AI value proposition directly
2. Requires the fewest dependencies on systems that don't exist yet
3. Produces the clearest user signal about whether the core concept works

For most AI MVPs, this is the moment the AI first responds to user input and produces something useful. Everything before that moment is setup. Everything after it is enhancement. Build to that moment as fast as possible, and let a real user experience it.

The Daily Build Protocol

Each day in Week 2 follows a four-part structure. Deviations from this structure are the leading cause of week-end status reports that say "mostly there" but can't demonstrate anything working.



The AI Layer: What to Build and What to Buy

One of the highest-stakes decisions in Week 2 is where your AI layer lives and what it does. The wrong answer here doesn't break the MVP — it makes the MVP expensive and fragile to maintain.

Three common patterns and their tradeoffs:

Pattern 1: Direct API Integration Your application calls OpenAI, Anthropic, or another provider directly. Simple, fast to build, lowest infrastructure overhead. Right for most MVPs. The risk: you are fully dependent on one provider's uptime, pricing, and model behaviour. Mitigate with retry logic and graceful error handling from day one.

Pattern 2: LLM Framework (LangChain, LlamaIndex) Adds orchestration, memory management, and tooling on top of the base model. Right when you need multi-step agent logic or complex retrieval. The risk: framework complexity can dramatically increase debugging time. Use only if the core use case genuinely requires it — most don't.

Pattern 3: Prompt Management Layer Even if you're using direct API calls, build a simple prompt management layer — a function or class that owns your prompts, handles versioning, and makes prompt updates deployable without code changes. This takes four hours to build in Week 2 and saves two days of pain in Week 4 when you inevitably need to update a prompt based on user feedback.

The Real Example: CaseFlow AI

Throughout this playbook, we'll follow a fictional but realistic example: CaseFlow AI, built for solo employment lawyers who need to draft client update emails from case notes.

Week 2, Day 8: The core loop — a text input for case notes, a button, and an AI-generated email draft — is running locally. It's ugly. It works.

Week 2, Day 9: The output is hosted on a real URL. Not deployed to production — just accessible on a staging environment. A real lawyer (founder's friend) tries it for the first time. She changes the subject line before sending. The team notes this.

Week 2, Day 10: Authentication is added. Not a full auth system — email and password, handled by a third-party library. Thirty minutes of work that would have taken two days to build from scratch.

Week 2, Day 11: The AI response adds a subject line suggestion. The prompt is refined based on five test cases. First version of the prompt management layer is in place.

Week 2, Day 12: Error handling. What happens when the API is down? What happens when the input is empty? What happens when the output is longer than expected? Each of these is a user experience decision, not just a technical one.

Week 2, Day 13: The lawyer tries it again with real case notes from an actual client file. She uses the output directly, with one edit. The founder watches without saying anything.

Week 2, Day 14: The Week 2 gate check. One complete user journey — log in, paste case notes, receive a draft email, copy it, send it — works end to end in a real environment. A real user has used it. There is signal.

Week 2 Gate Check: Has one real user (not a developer, not the founder) used the core AI feature and produced a clear signal — positive or negative — about whether it delivers the core value? If yes: enter Week 3. If no: stay in Week 2 until this is true.

Week 3: Harden

Days 15–21 · Mode: Break It on Purpose

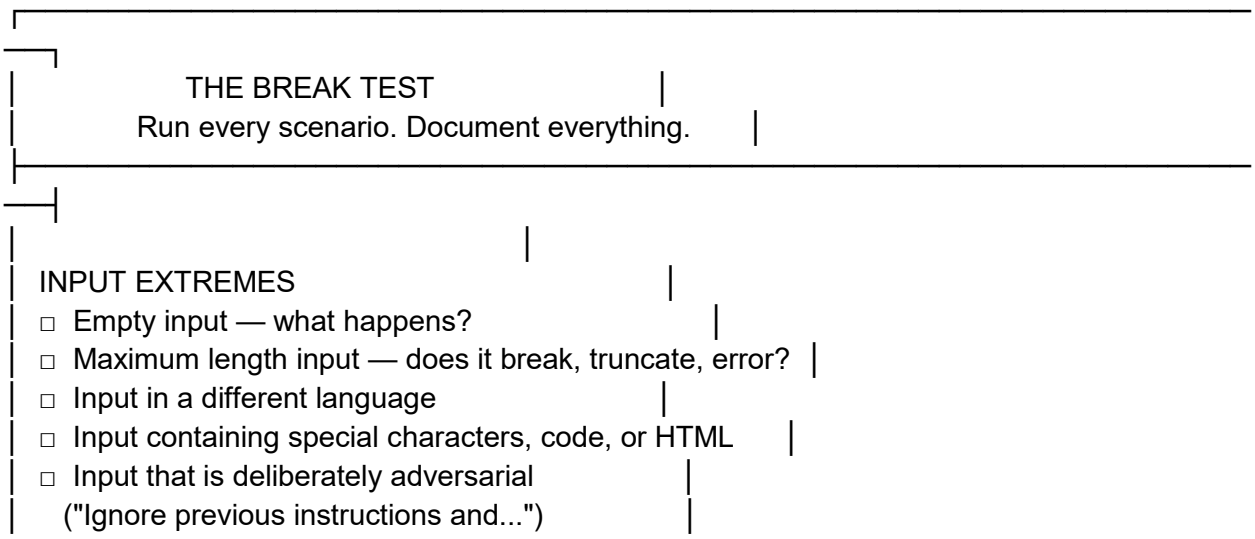
Week 2 builds the happy path. Week 3 is about everything that isn't the happy path.

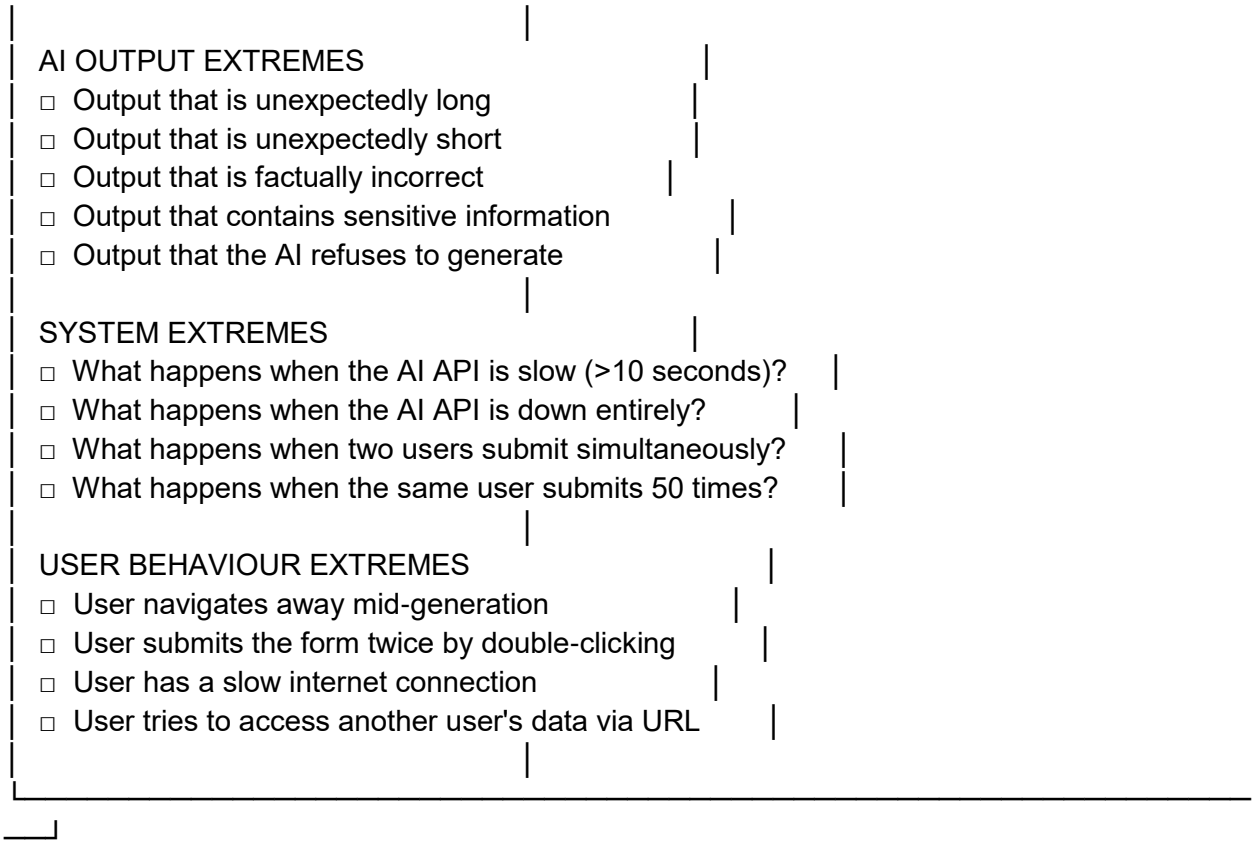
Most teams skip Week 3 or compress it to a day and a half. This is the decision that produces the majority of post-launch failures. The product works perfectly for the specific scenarios it was built for and behaves unexpectedly for everything else. At small user scale, this is embarrassing. At real user scale, it's a reputational and sometimes a legal problem.

Week 3 has three distinct jobs:

Job 1: The Break Test

The Break Test is a structured attempt to make the product fail. It is not QA. QA confirms that expected inputs produce expected outputs. The Break Test deliberately introduces unexpected inputs, unexpected load, unexpected sequences of events, and unexpected user behaviour.

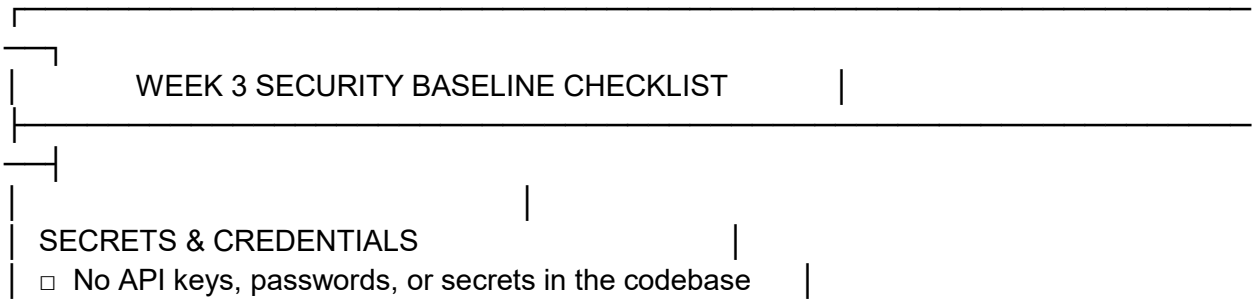




Every failure mode uncovered in the Break Test becomes a ticket. The tickets are triaged into three categories: must fix before launch, fix in week one post-launch, and known limitation to document. Anything that could expose user data, produce harmful output, or cause financial damage is a must-fix without exception.

Job 2: Security Baseline

AI MVPs built quickly — especially those using AI-assisted development tools — have a consistent and predictable set of security vulnerabilities. Running the following checklist is not optional. It is a 4-hour investment that prevents a category of problems that can end companies.



```
│ □ All secrets in environment variables                    │
│ □ .env files excluded from version control                │
│ □ API keys have minimum required permissions only         │
│                                                           │
│ AUTHENTICATION                                            │
│ □ Sessions expire at a reasonable interval                │
│ □ Password reset flow cannot be exploited                 │
│ □ Rate limiting on login attempts                         │
│                                                           │
│ DATA                                                      │
│ □ Users cannot access other users' data via URL           │
│   manipulation or API parameter changes                   │
│ □ User inputs are validated server-side, not just UI      │
│ □ AI outputs are not stored with PII unless required      │
│                                                           │
│ AI-SPECIFIC                                               │
│ □ System prompt is not accessible to end users            │
│ □ Prompt injection attempts are handled gracefully        │
│ □ Rate limiting on AI endpoint (prevents cost explosions) │
│ □ Maximum input length enforced                           │
│ □ Output length capped to prevent runaway responses       │
│                                                           │
│ TRANSPORT                                                 │
│ □ All traffic over HTTPS                                  │
│ □ No sensitive data in URL parameters                     │
│                                                           │
└───────────────────────────────────────────────────────────┘
```

Job 3: The Performance Baseline

Before launch, you need to know — with actual numbers — how your system performs. Not "it feels fast" but: what is the p95 response time for an AI generation? At what concurrent user count does response time degrade? How much does each AI call cost at the usage levels you're projecting?

These numbers matter for three reasons. First, they set the expectation for your monitoring alerts in Week 4 — you cannot alert on degradation unless you know what normal looks like. Second, they inform your infrastructure decisions before you're at scale rather than during a crisis. Third, they let you have an honest conversation with your first users about what to expect.

CaseFlow AI, Week 3 example:

The Break Test surfaces two must-fix issues: a double-submission bug that generates two emails when the user clicks the button quickly, and a prompt injection vulnerability where a user can override the tone instructions by including certain phrases in the case notes. Both are fixed in Day 16.

The security checklist reveals that API keys were in a configuration file that would have been included in the public repository. This is fixed in Day 16 before a single commit reaches the remote.

Performance baseline: p50 response time is 4.1 seconds. p95 is 9.8 seconds. This informs a UI decision — a progress indicator with estimated time — and a hard API timeout of 15 seconds with a helpful error message rather than an infinite spinner.

Week 3 Gate Check: The Break Test has been run in full. Every must-fix issue is resolved. The security checklist is complete with no open critical items. Performance baseline is documented. Enter Week 4.

Week 4: Ship

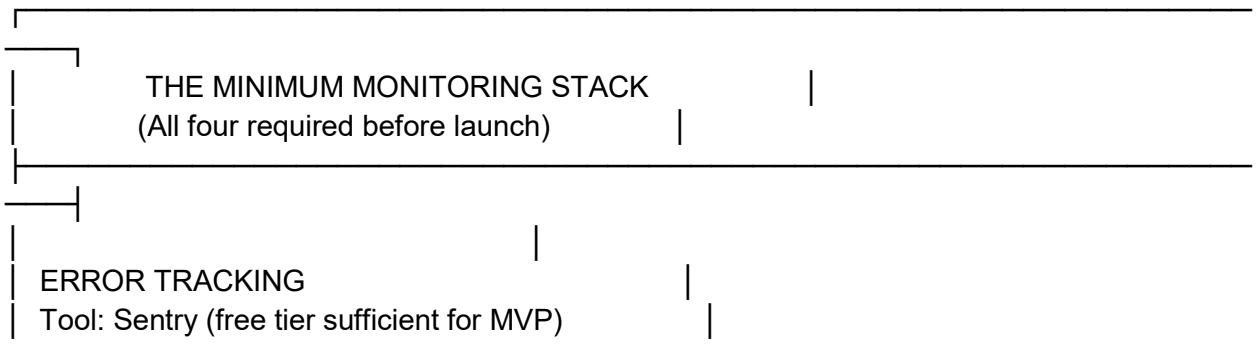
Days 22–28 · Mode: Release & Instrument

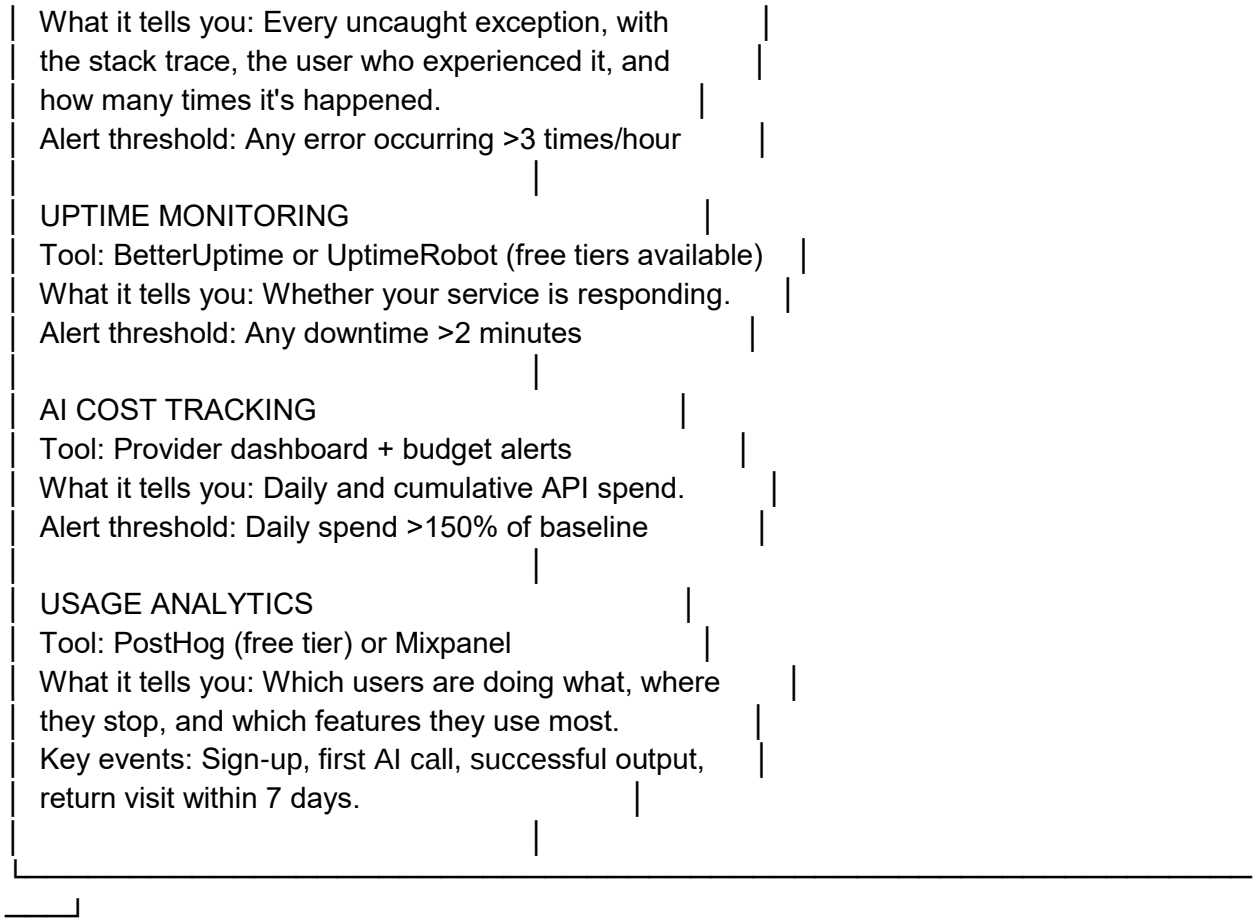
Week 4 is the week that most build guides dedicate entirely to launch logistics. It is not. Launching is one afternoon of Week 4. The more important work is what you put in place before and after that afternoon.

Days 22–24: The Monitoring Stack

You do not launch without monitoring. The monitoring stack is not a post-launch enhancement. It is a pre-launch prerequisite.

You need to know, in near real-time:





Beyond the technical monitoring stack, there is one instrument that most teams underinvest in during launch week: a feedback mechanism that is so frictionless users actually use it. This is not a five-question survey. It is a single-field text input, available on every screen, that asks: *"What's one thing about this that you'd change?"*

Day 25: The Controlled Launch

A controlled launch means your first real users are people you have chosen deliberately, not people you have acquired through marketing. Specifically:

Five to ten beta users who:

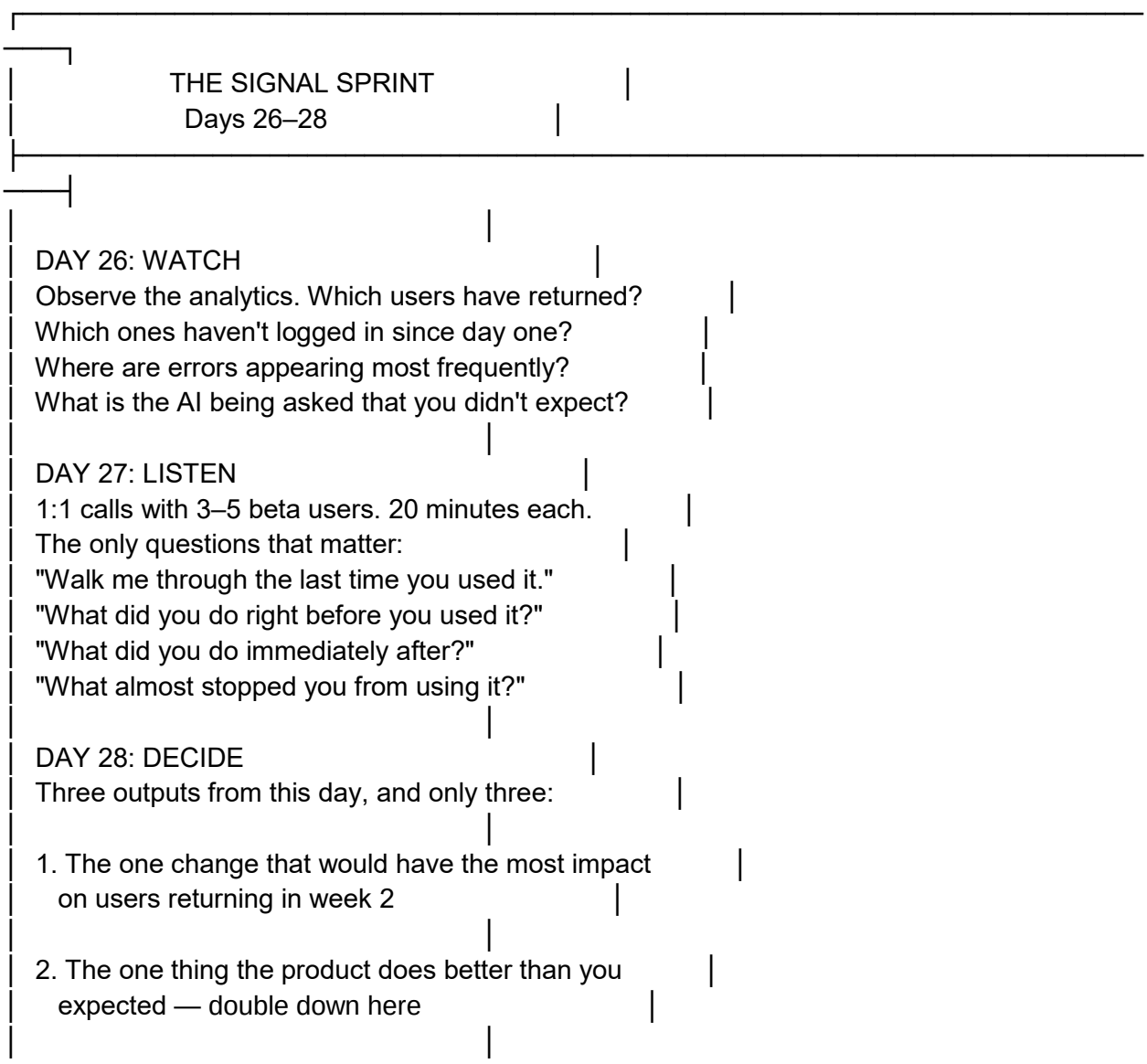
- Match your target user profile precisely
- Have the problem you're solving actively right now, not historically
- Have agreed in advance to give you 20 minutes of feedback in the first week
- Understand they're using an early version — set this expectation explicitly

This is not a soft launch strategy. It is an information strategy. Ten engaged users who you can talk to directly are worth more than a thousand cold signups in the first two weeks. They will tell you things that no analytics dashboard will surface: the workaround they built around your flow, the thing they tried to do that the product doesn't support, the reason they almost didn't sign up.

The controlled launch also protects you from the specific failure mode of discovering a critical bug at scale. Better to find it with ten users who you've already set expectations with than ten thousand users who you have not.

Days 26–28: The Signal Sprint

The last three days of the 28-day cycle are not building days. They are listening days.



```
│ 3. The one assumption in the One Thing Document          │
│    that was wrong — update it now for Week 5+            │
│                                                          │
└──────────────────────────────────────────────────────────┘
```

The Master Timeline: All 28 Days

WEEK 1 — DEFINE

```
Day 1  │ The problem audit. List every pain the target user has.
       │ Rank by: frequency × cost × solvability by AI.
Day 2  │ Draft the One Thing Document. First version only.
Day 3  │ One Thing Document finalised and signed off.
       │ The Not-List has at least three items.
Day 4  │ Technical direction decision. Which AI approach?
       │ Stack decision (framework, hosting, auth provider).
Day 5  │ Stack decision confirmed. Local dev environment running.
       │ First spike: can you make a raw AI call from your stack?
Day 6  │ Paper walkthrough. Every screen. Every AI call.
       │ Every error state. Draw it. Don't build it.
Day 7  │ Week 1 gate check. Resolve any gaps.
       │ Week 2 plan: day-by-day card list prepared.
```

WEEK 2 — BUILD

```
Day 8  │ The core loop. Input → AI call → output. Ugly. Working.
Day 9  │ Accessible on a real URL. Auth layer added (library, not
       │ custom). First real-user test. Document the confusion.
Day 10 │ Prompt refinement based on Day 9 feedback.
       │ Prompt management layer in place.
Day 11 │ Error handling for AI layer (API down, timeout, empty
       │ output). User-facing messages drafted.
Day 12 │ Secondary feature from the paper walkthrough — the one
       │ that users will immediately want after the core works.
Day 13 │ Second real-user test with updated product.
       │ Watch, don't explain. Document.
Day 14 │ Week 2 gate check. One complete journey, end to end.
       │ Clean up the worst of the technical debt. Commit.
```

WEEK 3 — HARDEN

Day 15 | Break Test — input extremes and AI output extremes.
| Triage all failures into must-fix / later / known.
Day 16 | Fix all must-fix Break Test items from Day 15.
Day 17 | Break Test — system extremes and user behaviour extremes.
| Security checklist: credentials, authentication, data.
Day 18 | Fix all must-fix items from Day 17.
| AI-specific security: prompt injection, rate limiting,
| output length caps.
Day 19 | Performance baseline. Load test core flows.
| Document p50, p95 response times for AI calls.
| UI adjustments based on actual latency numbers.
Day 20 | Staging environment that mirrors production exactly.
| Full end-to-end test in staging. Everything.
Day 21 | Week 3 gate check. Break Test complete. Security
| checklist complete. Monitoring stack configured.

WEEK 4 — SHIP

Day 22 | Monitoring stack live in production environment.
| Error tracking, uptime monitoring, cost alerts.
Day 23 | Usage analytics configured. Key events instrumented.
| Feedback mechanism live on every screen.
Day 24 | Final production deployment. Smoke test every
| user journey. Monitoring alerts tested.
Day 25 | Controlled launch to 5–10 beta users.
| Personal messages to each. Expectations set.
Day 26 | Watch the analytics. Error logs reviewed.
| Support any beta users who hit issues.
Day 27 | User calls. 3–5 users, 20 minutes each.
| The only three questions that matter.
Day 28 | Signal Sprint debrief. Three decisions only.
| Week 5 plan drafted based on real signal.

The Five Failure Modes — And How to Avoid Them

These are the specific points in the 28-day cycle where AI MVP builds most commonly derail. Knowing where they are in advance is the closest thing to a guarantee that you will make it to Day 28 with something real.

Failure Mode 1: Scope creep between Days 3 and 8 The Not-List in the One Thing Document starts getting violated. "Just one more feature" compounds across four days and Week 2 begins two features heavier than it should be. Prevention: the One Thing Document is read aloud at the start of every day in Week 1. If a proposed addition isn't in it, it goes on the Week 5 list without discussion.

Failure Mode 2: Architecture paralysis in Week 2 The team gets into a multi-day debate about the right database schema, the right framework, or the right deployment approach. Everything stops being more real. Prevention: the Ratchet Rule. Any architecture decision that prevents the day's card from being completed by 5pm is resolved by choosing the simpler option and moving on. You will not regret this.

Failure Mode 3: Skipping the real-user tests in Days 9 and 13 Teams convince themselves that getting an external person to test the product will slow them down. The opposite is true — every time a real user finds something unexpected, it saves days of building the wrong thing with high confidence. Prevention: book the Day 9 and Day 13 test users before Week 2 starts. Make the appointment harder to cancel than to keep.

Failure Mode 4: Compressing Week 3 to "a quick review" Week 3 is treated as optional polish time rather than structural hardening. The product launches with unresolved Break Test failures and no monitoring. The first real-world edge case becomes a crisis instead of a learning. Prevention: Week 3 has an explicit gate check. You cannot enter Week 4 without it. This is not a soft guideline.

Failure Mode 5: Measuring the wrong thing in Week 4 The team tracks signups and interprets any activity as validation. What you actually need to measure in Week 4 is return rate: do users come back after their first session? A product that users try once and don't return to has not been validated, regardless of how many people signed up. Prevention: the usage analytics from Day 22–23 are configured to track return visits within 7 days as the primary success metric before launch, not after.

A Note on What "MVP" Actually Means

The term minimum viable product has been so badly abused that it now means almost nothing. It has been used to justify shipping products with no error handling, products that only work for the founder's specific use case, and products that are "minimal" in every dimension including value.

The operative word in MVP is not minimum. It is viable.

Viable means it works reliably for a real user doing a real task. It handles the obvious failure modes gracefully. It doesn't expose their data. It doesn't fall over when two people use it simultaneously.

Minimum refers to the scope — the number of problems it solves, the number of user journeys it supports, the breadth of the feature set. The scope should be ruthlessly minimal. The quality of execution within that scope should not be.

This distinction is why Week 3 exists. Minimum scope, viable quality. Every day of Week 3 is in service of the second word, not the first.

After Day 28: The Honest Conversation

You will reach Day 28 with something real. Real users have used it. You have signal — numbers and conversations — that tell you whether the core value proposition works.

What you will not have is a finished product. This is correct. The 28-day cycle is designed to get you to the most valuable position as quickly as possible: a working system, in the hands of real users, with enough instrumentation to know what to build next.

The question at the end of Day 28 is not "is it done?" It is: "what did we learn that we couldn't have learned any other way?"

The answer to that question, extracted from real user behaviour and direct conversations, is what determines whether Week 5 is an accelerated build cycle, a strategic pivot, or the most informed decision to stop you could possibly make.

All three of those outcomes are better than six months of building without signal.

How Susea.ai Uses This Playbook

This is not a theoretical framework. It is the operating system behind every Rapid MVP engagement we run.

The 4-week guarantee we offer clients is built on every component of this playbook: the One Thing Document that prevents scope creep, the daily build protocol that creates accountability, the Break Test and security baseline that prevent post-launch disasters, and the Signal Sprint that means clients leave Day 28 with real user data, not just a deployed URL.

The difference between a 4-week MVP that generates real traction and a 4-week MVP that gets shelved in month two almost never comes down to the idea or the technology. It comes down to whether the team building it has a system that prevents the five failure modes above.

That system is what we bring. The idea is what you bring.

If you have an AI product idea and want to run this playbook with an experienced engineering team — or if you'd like a 20-minute review of your current build plan before you start the clock — our team is available at susea.ai

We'll tell you whether your scope is right for 28 days, which technical approach is appropriate for your use case, and where in the five failure modes you're most at risk.

The conversation is free. The cost of not having it is usually about 6 weeks.

Susea.ai *Fix · Build · Deliver From idea to production in 28 days. Every time.*
