



HARRIS

The Afternoon Build Kit

13:30 → 16:00

The four working papers behind one system that went from a written spec to live and self-running in a single afternoon. With the blanks marked, so you can use them on yours.

Five moves, in this order.

The afternoon worked because of the order things were done in, not because of speed. Each paper in this kit belongs to one of these moves.

01 Write the spec as if the builder cannot ask you anything.

Paper one. Say what done looks like, what must not be built, and how success is tested. A spec that only describes the idea produces a plan, not a system.

02 Review it against reality before any code is written.

Paper two. Fifteen minutes spent checking what the spec assumes is cheaper than three hours building on something that isn't there.

03 Cut to one loop that works end to end today.

Roughly half of any good spec is hardening for a loop that hasn't run once. Build the loop. Harden what the first runs prove needs it.

04 Verify in the real system, not in the report.

Paper three. "It works" means you looked at the place where it would be broken, not that the tool said so.

05 Write down what broke the same day.

Paper four. A failure that is logged with its cause is a test you get to keep. One that isn't comes back.

2 h 30

SPEC TO FIRST PUBLIC
POST

54

AUTOMATED TESTS BY
EVENING

8

FAILURES CAUGHT AND
LOGGED

The build prompt, as a template.

This is the structure of the spec that started the afternoon. Replace anything like this . Keep every section, even when the answer is one line.

1. Mission

Build `name` , a `one-sentence description` . This is phase one only. The objective is to get `the complete loop` working reliably, with `no routine human step` .

Why: one objective, stated as a loop, is what makes "done" checkable.

2. What already exists

Start from `existing tool or repository` . Before building, inspect what it actually does, its licence and what it costs to run. Reuse it. Do not rewrite it.

Why: this instruction is what caught the wrong assumption on the day.

3. The workflow, as numbered steps a user takes

`1. I open... 2. I add... 3. It begins... 4. When X happens... 5. It records...`

Why: steps written from the user's side expose missing parts faster than a feature list.

4. Limits

Do not build `the five things you'll be tempted to add` . These are future modules.

Why: a builder with no "do not" list will fill the day with infrastructure.

5. Reliability, in plain rules

A failed job must never block unrelated work. Never report success for something that did not complete. A retry must not create a duplicate.

6. Acceptance tests

Do not call it complete because the screen loads. Test: `set-up persists · the trigger fires once · real output is produced · a real action reaches the outside world · a retry is safe · a restart recovers · the full unattended run` . Where a test is blocked, say exactly which and why. Never invent a result.

7. The closing lines

Make ordinary technical decisions without asking. Do not stop after a plan or a mock-up: build the working thing. If part of it proves impossible, say precisely what blocked it and finish the rest.

The eight questions asked before building.

Run these against the spec, and answer each from the real thing: the actual code, the live documentation, the machine it will run on. In green: what each one turned up that afternoon.

01 Does the thing we're building on actually do what the spec assumes?

Read its code and its own description, not a summary of it.

Found: the engine could clip, but could not watch other creators or post through the official interface. Both had to be built.

02 What does its licence allow?

Especially if this might ever be sold or offered to others.

Found: one candidate carried a licence that ruled it out for anything commercial; part of the other was restricted too.

03 What does the platform's rule say today, not last year?

Check the live page and its date.

Found: an upload restriction everyone still warns about had been lifted four days earlier.

04 Where does the spec contradict itself?

Look for two sentences that cannot both be obeyed.

Found: "approved items only" in one section, "no human step" in another.

05 Can this machine run it?

Check the hardware, the installed tools and the free space before the build starts.

06 Which steps can only a human do?

Sign-ins, consents, payments. List them now so they don't stop the build at five to four.

07 What is the smallest loop that could run end to end today?

Name it, and name what is being left for later.

08 How does this earn, or what does it prove?

If nobody can answer, say so out loud before building. Then decide anyway, knowingly.

Checked, not assumed.

The list behind every "it works" that day. Each line exists because skipping it cost something, that afternoon or before.

- ☐ **Confirm in the place it would be broken.**
Not in the tool's reply. A post is published when the platform shows it finished, not when the upload call returns.
- ☐ **A control that looked clicked may not have been.**
After any setting change, reload the page and read the setting back.
- ☐ **An error is not proof of failure, and an entry is not proof of success.**
Look for the real result before retrying, and check it is complete before believing it.
- ☐ **Before any retry, ask: did the first attempt partly land?**
Give every outside action a marker you can search for.
- ☐ **Test the first real action privately.**
The first upload went to the wrong place. It was private, so it cost nothing.
- ☐ **Restart it on purpose.**
If state only lives in memory, you find out now or at three in the morning.
- ☐ **Test the refusal, not just the success.**
Send the request without the key. Send the malformed input. Confirm it says no.
- ☐ **One failure must not stop the queue.**
Break one item deliberately and confirm the others still move.
- ☐ **Say what was not verified.**
"Built, tested against a stand-in, not yet run against the live service" is a complete and honest status.
- ☐ **Never let a secret pass through a screen or a log.**
Move it from file to file. If one does leak, say so and rotate it the same day.

The failure log.

Eight things that broke between 13:30 and midnight, in order. None of them was dramatic. All of them would have been, left alone.

WHAT BROKE	HOW IT WAS CAUGHT	WHAT CHANGED
The spec assumed the engine did more than it does	Reading the engine's code before writing ours	Two missing parts built; the plan cut to one loop
A setting that looked enabled wasn't	Re-opening the page and reading its status	Every setting change is now read back
The first upload went to the wrong channel	It was sent private, and the channel name was checked	The destination is confirmed before anything public
Creators added, nothing clipped	Comparing what the feed held with what was stored	A different feed, and a way to include recent uploads
One of three stored keys was dead	The service's own error message	The working key wired in; the dead one flagged
The title writer returned nothing for some clips	Reading the failure messages on the dashboard	A larger reply allowance and a stricter parser
An upload was reported as failed	Looking at the channel itself	Errors are now checked against the channel first
The same upload, marked published, was an empty entry	The founder noticed zero views and no thumbnail	Nothing counts as published until the platform has finished processing it

The last two are the same event, misread twice in opposite directions. The first fix was confident and wrong. That is why the list on the previous page says "an entry is not proof of success".

Harris is becoming a product.

You're on the waitlist, so you'll get one email when it opens. Until then: pick the thing you've been planning, and start at paper one.