From Prompt to System

From Prompt to System

A Practical Guide to Building Useful AI Workflows

How to move from an AI idea to a responsible working system.

Crate South Education · Corey Ealy

> **How to use this guide.** Read it front to back once. Then keep it on the desk: the worksheets, canvas, and checklists at the back are designed to be photocopied, projected, and used in class or in a working session. Everything here is public educational material — no proprietary systems, no client data, no secrets. The method is the product.

Chapter 1 — Why prompting is only the beginning

Almost everyone meets AI the same way: a text box, a question, an answer that arrives suspiciously fast. It feels like the whole thing. It is not the whole thing.

A prompt is a conversation with a model. A **system** is what happens when that conversation is placed inside a real workflow — with inputs that arrive on time, decisions someone is accountable for, a human who approves what matters, actions that leave a record, and measurements that tell you whether any of it helped.

Here is the difference in one example. A small business owner asks a chatbot, "Write a friendly reply to a customer who missed us." Fine — a decent paragraph appears. But who sends it? When? To which customers? What does it cost per month? What happens when the reply is wrong? Who approved offering a discount it invented?

Nothing about the prompt answers those questions. Those questions *are* the system.

This guide teaches you to design that system. Not in the abstract — on paper, with worksheets, against real problems. By the end you will be able to take an idea like "we should use AI for follow-ups" and turn it into a mapped, bounded, measurable workflow that a builder could implement and an owner could defend.

The one-sentence version: AI literacy is not knowing what to type into the box. It is knowing what the box is for, what it must never do, and who answers for it.

Reflection. Think of the last time you used an AI tool for something real. What happened before you opened it? What happened to the output after? Who else was affected? That before-and-after is the system you were already part of — whether anyone designed it or not.

Chapter 2 — Start with the problem

Every useful system starts with a problem experienced by specific people. Every useless one starts with a tool looking for something to do.

A good problem statement has three parts:

1. **Who** is affected — named people, not "users."
2. **What happens today** — the current reality, described honestly.
3. **What it costs** — in time, money, missed opportunity, or frustration.

Compare:

- × "We should use AI for customer service." (A tool, not a problem.)
- ✓ "Our office manager misses 15–20 calls a week while on job sites. Most callers don't leave voicemail — they call the next shop on the list. Each lost call is a lost job worth a few hundred dollars." (Who, what, cost.)

The second statement tells you what to build, what to measure, and when to stop. The first tells you nothing.

Testing your problem statement. Read it to someone who lives the problem. If they nod and add detail, you have a real one. If they shrug, you have a guess.

Common failure modes:

- *Solution-shaped problems.* "We need a chatbot" is a solution wearing a problem's clothes. Ask: what would be true if the chatbot worked perfectly? That answer is closer to the problem.
- *Everything problems.* "Improve communication" fits any fix and guides none. Narrow until it stings a little.
- *Assumed problems.* You are not the user. The fastest way to waste a build is solving a problem nobody confirmed.

Exercise 2.1 — Name it. Write your problem in the three-part form: who, what happens today, what it costs. Then cut every adjective. If it still hurts, keep it.

Chapter 3 — Understand the people and the context

Systems don't run in labs. They run in buildings with budgets, habits, short staffing, old laptops, strong opinions, and rules. The people and the context are not scenery — they are most of the design.

Map four groups of people:

1. **The served** — who benefits when it works (customers, students, applicants).
2. **The operators** — who runs it day to day (reviews drafts, handles exceptions).
3. **The touched** — whose data or experience passes through it, whether or not they ever see it (the person whose voicemail gets transcribed).
4. **The accountable** — the named human who answers when something goes wrong. If this seat is empty, stop. You are not ready to build.

Then list the context honestly:

- Budget — not what you wish, what exists.
- Tools already in use — people trust what they know; plan to meet them there.
- Skills — who will operate this, and what can they realistically learn?
- Constraints — policies, laws, funding rules, union agreements, parental consent.
- Environment — connectivity, devices, shared machines, noise, time pressure.

The context test. Describe your imagined system to the most skeptical person who would have to live with it. Whatever they object to first is your real context list.

> **Exercise 3.1 — The four groups.** For your problem, name one real person (or role) for each group: served, operator, touched, accountable. If any seat is empty, write down who *should* fill it — that gap is now on your map.
>
> **Reflection.** Which group is easiest to forget? (It's almost always "the touched" — people whose data moves through systems they never agreed to.)

Chapter 4 — Map the workflow before choosing tools

This is the discipline that separates builders from buyers: **draw the steps before you pick the software.**

A workflow map answers, in order:

1. **Trigger** — what starts it? (A missed call. A submitted form. A date arriving.)
2. **Steps** — what happens next, and next, in plain verbs. "Log the caller's number." "Check the schedule." "Draft a reply."
3. **Ownership** — who or what does each step.
4. **Waiting** — where does work sit? (The unreviewed draft. The unanswered email.) Waiting is where workflows go to die.
5. **Exceptions** — the angry caller, the missing form, the holiday, the typo. List the top three. Systems are judged by their worst day, not their best.

Draw it with boxes and arrows on paper. If you can't draw it, you don't understand it yet — and no tool will fix that.

A mapping session that works: put the person who actually does the work in the room. Ask them to describe last Tuesday, not the official process. The map is wrong until they say it's right.

Where AI fits — and where it doesn't. Only after the map exists do you ask: which of these steps could AI help with? Good candidates are narrow and checkable: draft, summarize, classify, extract, transcribe, translate. Bad candidates are vague ("handle customer service") or consequential ("decide who's eligible").

> **Exercise 4.1 — Last Tuesday.** Map a workflow you know well as it actually happened most recently — every step, every wait, every handoff. Mark each step: human, tool, or either. Circle the steps where work waited. Those circles are your design targets.

Chapter 5 — What AI should and should not do

Here is the working rule set. It's short enough to memorize and strict enough to keep you out of trouble.

AI is good at:

- **Drafting** — first versions a human will review (replies, summaries, agendas).
- **Summarizing** — long to short (meeting notes, applications, documents).
- **Classifying** — sorting into buckets you defined (urgent/routine, topic, intent).
- **Extracting** — pulling structured facts from messy text (names, dates, amounts).
- **Translating** — between languages and between jargons (technical ↔ plain).

AI is bad at — or must not do alone:

- **Consequential decisions** — eligibility, discipline, hiring, pricing, safety. AI may prepare information; a human decides.
- **Knowing your facts** — your prices, policies, and schedules come from your owned knowledge (Chapter 6), not from the model's memory.
- **Caring about outcomes** — the model has no stake. People do.
- **Knowing when it's wrong** — it will be confident either way. Your checks, not its confidence, decide what ships.

The one-sentence test for any AI job: "Could a careful person check this output in under a minute?" If yes, it's probably a good AI job. If checking takes as long as doing, skip it.

> **Exercise 5.1 — Sort the jobs.** List every task in your mapped workflow. Sort each into: *AI drafts/assists*, *human only*, or *undecided*. For everything undecided, apply the one-sentence test and move it.

Chapter 6 — Inputs, decisions, approvals, and actions

These four are the load-bearing walls of any AI system. Get them right and almost everything else follows.

Inputs — what the system is allowed to know. Define exactly which inputs the system receives, from whom, in what format, and with what permission. Three rules:

1. Collect only what the purpose requires.
2. Personal information needs a permission trail.
3. Assume inputs will be late, incomplete, or wrong — and design for it (ask again, flag, or route to a human instead of guessing).

Decisions — where choices change outcomes. Walk your workflow map and mark every point where a choice matters. For each one, write: what is being decided, what the criteria are, and who or what may decide it. The decision nobody noticed is the one that gets you in trouble — automated systems make hidden decisions constantly unless you drag them into the open.

Approvals — a named human says yes. Before anything consequential happens — sending, spending, publishing, enrolling, declining — a person reviews and approves. Design approvals to be *fast to review, easy to correct, and impossible to skip silently*. A good approval shows the human exactly what will happen, in the output's final form, with one tap to approve and one to edit.

Actions — touching the real world. Actions are messages sent, bookings made, records changed, money moved. For each action your system can take, write down: is it reversible? is it logged? what limits apply (volume, timing, recipients)? The bigger the blast radius, the tighter the control. Reversible + logged + limited actions can run with light review. Irreversible ones get the strongest gate you have.

> **Exercise 6.1 — The four walls.** For your workflow: list the inputs (with permission status), mark the decision points (with criteria and decider), name the approver for anything outbound, and rate every action reversible/irreversible. Any wall you can't complete is your next conversation, not your next build.

Chapter 7 — Human-in-the-loop design

"Human in the loop" is often said as a slogan. Here is what it means as engineering.

Decide where the human sits. There are three honest patterns:

1. **Review before action** — AI drafts, human approves, then it happens. Use for anything outbound or consequential.
2. **Review after action, with rollback** — system acts, human audits a sample or gets flagged exceptions. Use only for reversible, low-stakes actions, with real sampling.
3. **Human acts, AI advises** — the human does the work; AI organizes, summarizes, and prepares. Use wherever stakes are high or trust is still being built.

Design the review interface, not just the model. Reviewers approve what's easy to approve. If checking a draft takes ten minutes, reviewers rubber-stamp — and your "human in the loop" is a rumor. Show the draft in final form. Show what changed. Make editing one tap. Make the queue short enough to stay honest.

Watch the failure modes:

- *Rubber-stamping* — queue too long, stakes feel low. Fix: shorter queues, spot audits, escalate the weird ones.

- *Alert fatigue* — everything flagged is nothing flagged. Fix: flag less, mean it more.
- *Skill decay* — the human approves so long they forget how to do the task. Fix: regular no-AI drills for critical tasks.

> **Exercise 7.1 — Place the human.** For each AI job in your workflow, choose one of the three patterns and write why. If you chose pattern 2 anywhere, write the rollback procedure. If you can't, change it to pattern 1.

Chapter 8 — Boundaries and failure conditions

Before launch, write the **never-list**: what the system must never say, decide, send, spend, or store. Then design the safeguard that enforces each line, and test that it fires.

A missed-call reply system, for example:

Never	Safeguard	Test
Quote exact prices	Drafts use price ranges only; approval checklist	Send test inputs asking for prices; confirm range-only output
Message opted-out contacts	Opt-out list checked before every send	Opt a test number out; attempt a send; confirm block
Exceed monthly budget	Hard spend cap with alert	Simulate cap; confirm halt + alert

Failure conditions deserve their own list. Ask of every stage: *what's the worst plausible thing that happens here?* Inputs arrive garbled. The knowledge sheet is six months stale. The reviewer is on vacation for two weeks. The model confidently summarizes a voicemail about a gas leak as a routine callback. For each failure, decide: prevent it, detect it, or absorb it — and write down which.

The stop procedure. Every system needs a kill switch: a named person, a defined trigger ("if X happens, stop"), and a way to halt that a tired operator can execute in under five minutes. If you can't describe how to stop it, you haven't finished designing it.

> **Exercise 8.1 — The never-list.** Write five lines your system must never cross. For each: the safeguard, and how you'd test it this week. Then write the stop procedure in three steps.

Chapter 9 — Prototyping quickly

A prototype is a cheap way to be wrong before it gets expensive.

Order of operations:

1. **Paper first.** Walk the workflow map with real example inputs. Where does it break? Paper breaks are free.
2. **Wizard-of-oz second.** Run the workflow with a human secretly playing the AI's role: you draft the replies by hand and route them through the approval step. You learn the real volumes, the awkward cases, and whether the approver's job is even possible — before writing a line of code.
3. **Thin slice third.** Automate one step — just one — for a handful of real cases. Measure it against the baseline. Expand only what earns it.

What a prototype must answer:

- Do the inputs actually arrive the way the map assumed?
- Is the AI output good enough to review quickly, most of the time?
- Does the human approval step hold up under real volume?
- Do people actually want the output? (The fastest way to learn this is to watch someone receive it.)

What a prototype is not: a demo built to impress. A prototype is built to *fail informatively*. If nothing about your design changed during prototyping, you weren't testing — you were performing.

> **Exercise 9.1 — Wizard week.** Pick one workflow. For five real cases, play the AI's role yourself, by hand, through the approval step. Log: time per case, edits the approver made, cases you couldn't handle. That log is your build spec.

Chapter 10 — Measuring usefulness

If you can't say how you'll know the system is failing, you're running on hope.

Choose few, honest metrics. Three to five, in three families:

- **Speed** — response time, time-to-complete. (Missed calls answered within the hour: percent.)
- **Quality** — error rate, edit rate on drafts, complaint rate. (Drafts approved without edits: percent — and watch it honestly, not hopefully.)
- **Outcome** — the thing the problem statement cared about. (Missed calls that become booked jobs: percent.)

Take a baseline first. Measure the current reality before the system exists — even crudely, for two weeks. Without a baseline, every later number is a story you tell yourself.

Count outcomes, not activity. "Messages sent" is activity. "Problems solved" is outcome. Activity metrics are easy to inflate and correlate loosely with value; outcome metrics keep everyone honest.

Set the review rhythm. Metrics nobody reads are decoration. Pick a cadence — weekly for new systems, monthly once stable — with a named person who reads the numbers and can act on them.

Exercise 10.1 — The honest five. Write up to five metrics for your system: one speed, two quality, two outcome. For each: the target, the baseline (or "need to measure"), and where the number will live.

Chapter 11 — Cost awareness

AI systems spend three currencies: **money** (per-call fees, subscriptions, tooling), **time** (human review, maintenance, fixes), and **attention** (every notification and queue is a tax on a person).

Compute cost per action. Add up a month of the system's spending — all three currencies — and divide by the actions it completed. A follow-up text that costs $0.03 in model fees but two minutes of owner review time has a real cost closer to a dollar. Both numbers matter; the second one is the one people forget.

Compare honestly to value. What is one recovered customer worth? One staff hour returned? One applicant who doesn't give up? If cost per action exceeds value per action, you don't have a system — you have a hobby. (Hobbies are fine. Name them correctly.)

Watch the quiet growth. Usage creeps: more messages, longer documents, an extra "quick summary" feature. Set a budget alert at 80% of monthly cap, and review the bill the way you'd review any vendor invoice.

The build/no-build line. Sometimes the honest answer is: the problem is real, the workflow is mappable, and the math doesn't work — the volume is too low or the review time too high. Saying "don't build this" is a success of the method, not a failure of the idea.

Exercise 11.1 — The real number. Estimate your system's monthly cost in all three currencies, the cost per action, and the value per action. Write the ratio. Would you sign this as an invoice? Why not?

Chapter 12 — Documentation and evidence

A system nobody can explain is a system nobody can govern — including you, six months from now.

The minimum viable documentation set:

1. **The problem statement** — who, what, cost (Chapter 2).
2. **The workflow map** — current version, dated.
3. **The AI job list** — what the model does, one sentence each.
4. **The approval design** — who approves what, how fast.
5. **The never-list and safeguards** — with test results.
6. **The metrics** — what you watch, targets, baseline.
7. **The change log** — what changed, when, why, and what happened after.

Why evidence matters beyond tidiness. When a board member, funder, parent, or reporter asks "how does this thing decide?" — the answer should be a folder, not a shrug. Evidence is what lets other people trust your system without trusting your personality.

Keep it alive. Documentation that isn't updated is misinformation. Attach the review to a rhythm that already exists: the monthly metrics review updates the docs, or the docs get read aloud at the quarterly meeting.

> **Exercise 12.1 — The folder.** Create the seven-item set for your system, even if some items are one line long. Date each one. Put the review date on the calendar.

Chapter 13 — Improving the system

Launch is the middle of the story. Systems that matter get better on purpose.

The improvement loop: evidence → hypothesis → small change → measurement → keep or revert. Every change follows it, no exceptions for "obvious wins."

Where improvements come from:

- **Reviewer edits** — the corrections approvers make to AI drafts are the highest-quality training signal you own. If reviewers always delete the same sentence, stop writing that sentence.
- **Failure reviews** — every logged failure gets five minutes: what stage, what cause, what change would have caught it?
- **Recipient feedback** — complaints, compliments, and silence all count. Silence usually means your message didn't matter to them.
- **Metric drift** — numbers that slide slowly are systems telling you something changed in the world.

Rules for changing a running system:

1. One change at a time, or you won't know what worked.
2. Test before live, even if the test is ten manual cases.
3. Log every change with the reason.
4. Be able to revert. If a change can't be undone in minutes, it needs a stronger review before it ships.

> **Exercise 13.1 — The loop.** From your wizard-week log (Exercise 9.1), pick the single most common edit or failure. Write the hypothesis, the smallest change that tests it, and the metric that will tell you it worked.

Chapter 14 — Four public use cases, end to end

Each case below walks the full system map: problem → people → context → inputs → workflow → knowledge → AI jobs → decisions → approval → action → output → mea-

surement → feedback → improvement → boundaries → cost. All four are sanitized teaching composites — no real client, no real data.

14.1 Local business missed-call recovery

Problem. A home-services shop misses 15–20 calls weekly while the crew is on jobs. Most callers don't leave voicemail; they dial the next shop.

People. Callers (served); office manager (operator); callers again (touched — their numbers and voicemails); the owner (accountable).

Context. Small budget, one phone line, staff who live on phones, customers who expect a reply within the hour.

Inputs. Missed-call notifications (number, time), voicemail transcriptions, the price-and-services sheet, the weekly schedule. Permission: callers volunteered their number by calling; replies are service-related only.

Workflow. Call missed → notify → log caller → draft reply → manager reviews → approved text sent → outcome logged → reminder if no reply.

Knowledge. Services and price ranges, service area, appointment windows, the ten questions every caller asks. Owner: the office manager; review monthly.

AI jobs. Draft a short friendly reply referencing the likely need; summarize voicemails into three lines. Both checkable in under a minute.

Decision points. Routine vs. urgent (leak, no-cool in July). Urgent → alert the owner immediately; routine → draft a reply. Criteria written down.

Approval. The manager reads every draft — approve or edit. Nothing sends without it.

Action. Send approved text from the business number; create callback task for urgent. Reversible (a correction text), logged, rate-limited.

Output. A plain, warm message: thanks, how we can help, how to reach a person.

Measurement. Response time; missed calls → conversations; conversations → booked jobs; monthly cost. Baseline: two weeks of call logs before anything changes.

Feedback. Manager edits logged; "stop" honored instantly; complaints reach the owner same day.

Improvement. Fifteen minutes weekly: edit patterns, missing knowledge, response-time trend.

Boundaries. No exact prices by text. No marketing to missed callers. No contact after opt-out. Monthly spend cap.

Cost. Tooling + per-message drafting $\approx$ the value of one recovered job per quarter. Verified monthly by the owner.

14.2 Nonprofit volunteer intake

Problem. Applications sit for days; would-be volunteers move on. Intake eats staff hours a two-person team doesn't have.

People. Applicants (served); volunteer coordinator (operator); applicants' references (touched); executive director (accountable).

Context. Grant-funded, board reporting, some roles involve minors — screening is non-negotiable.

Inputs. Short application form, references for sensitive roles, calendar of needs. Permission: applicants consented on the form; reference data handled per policy.

Workflow. Apply → organize → acknowledge → summarize for staff → staff decide → notify → schedule orientation → log.

Knowledge. Role descriptions, screening requirements, orientation dates, FAQ. Owner: coordinator; review quarterly.

AI jobs. Draft acknowledgments; summarize applications into a standard brief; suggest interest-to-role matches. Suggestions only.

Decision points. Fit for a role; screening required. Staff decide every placement. No auto-accept, no auto-reject — ever.

Approval. Coordinator approves every message and every match before anything sends.

Action. Send approved messages; schedule orientation; send declined applicants a kind note with other ways to help.

Output. Prompt acknowledgment, clear next steps, a named human contact.

Measurement. Application → first response time; applicants reaching orientation; coordinator hours saved. Baseline: last quarter's inbox timestamps.

Feedback. One-question link after orientation; summary corrections logged; annual volunteer survey.

Improvement. Monthly stall-point review; summary format tuned from corrections.

Boundaries. Never skip screening. Never message minors without guardian contact. Reference data never stored beyond policy.

Cost. Low per application; the real saving is coordinator hours — from hours to minutes weekly.

14.3 Student project planning assistant

Problem. Teams lose week one of a four-week project to confusion: what are we building, who does what, when is it due?

People. Students (served); teacher (operator and accountable); teammates (touched — one person's delay is everyone's grade).

Context. Shared devices, 45-minute periods, mixed skill levels, academic honesty rules, one teacher for many teams.

Inputs. The assignment brief, the rubric, team skills and interests, checkpoint calendar. No personal data beyond class records.

Workflow. Read brief → answer structured planning questions → draft plan → team debates and edits → teacher approves → weekly tracking → end-of-project reflection.

Knowledge. Assignment requirements, example milestones, plain-language project terms. Owner: the teacher; review each semester.

AI jobs. Ask clarifying questions; draft a milestone plan from the team's answers; restate the rubric in plain language. It never writes project content — that's the honesty line, stated to students as a feature.

Decision points. Is the scope realistic? Team decides with the teacher; AI flags overloaded plans, nothing more.

Approval. Teacher approves every plan at kickoff and any major scope change.

Action. Approved plan becomes the team's visible task board; checkpoint reminders.

Output. A one-page plan every member can explain: goal, milestones, owners, dates, definition of done.

Measurement. Plans approved in week one; checkpoint completion; teacher time per team; can every member explain the plan?

Feedback. Two-minute weekly team check-in; end-of-project reflections.

Improvement. Teacher tunes the planning questions each semester from where plans broke.

Boundaries. Never do the assignment. Never message students outside class tools. Never compare teams to each other.

Cost. Runs inside existing class tooling; per-team cost small; teacher setup time is the real investment — budget two hours the first time.

14.4 Community event communications

Problem. Event information goes out late, inconsistent, or not at all — one volunteer juggling flyers, texts, and posts alone.

People. Attendees (served); communications volunteer (operator); partner organizations and attendees (touched); the board (accountable).

Context. Volunteers with day jobs; audiences split across text, email, social, and print; accessibility needs including plain language.

Inputs. Confirmed event details, channel contact lists, tone guidelines. Permission: every list is opt-in; opt-outs honored same day.

Workflow. Details confirmed → message calendar drafted → channel versions prepared → event lead approves each → scheduled sends → day-of updates same way → recap logged.

Knowledge. Past recaps, venue details, partner descriptions, standard answers (parking, accessibility, weather, cost, kids). Owner: event lead; review per event.

AI jobs. Draft channel-specific versions from one confirmed fact sheet; rewrite anything failing the plain-language check.

Decision points. What goes out when, to whom — the event lead decides the calendar. AI drafts against it; nothing improvises.

Approval. Every public message read and approved by the event lead. No exceptions, including day-of changes.

Action. Schedule and send through normal channels. Logged, reversible (correction message), volume-limited.

Output. Clear, consistent information in the organization's voice, with a named contact.

Measurement. On-time rate; attendance vs. invitations; questions the messages failed to answer; volunteer hours.

Feedback. Attendee replies logged; volunteer notes on heavy-edit drafts.

Improvement. Fifteen-minute debrief after each event updates the fact-sheet template.

Boundaries. Never send unconfirmed details. Never message opt-outs. Never post anyone's personal contact info.

Cost. A few dollars per event in drafting; the saving is volunteer hours — the scarcest resource the organization has.

> **Exercise 14.1 — Steal the structure.** Pick the use case closest to your world. Rewrite it for your organization, stage by stage. Wherever you write "it depends," that's your next conversation.

Chapter 15 — A practical build worksheet

Use this in one sitting, 60–90 minutes, ideally with the people who live the problem. A printable version lives in `templates/BUILD_WORKSHEET.md`.

1. The problem (10 min). Who is affected? What happens today? What does it cost? Write it in three sentences, no adjectives.

2. The people (10 min). Served / operators / touched / accountable — one name or role each. Empty seats are findings, not failures.

3. The context (10 min). Budget, tools in use, operator skill, rules, environment. Write the three constraints most likely to kill the project.

4. The workflow (15 min). Trigger → steps → waits → top three exceptions. Boxes and arrows. The person who does the work says when it's right.

5. The AI jobs (10 min). From the map: which steps could AI draft, summarize, classify, extract, or translate? One sentence per job. Apply the checkable-in-a- minute test.

6. The human design (15 min). Decision points + criteria + decider. Approval gate: who, how fast, reviewing what. The never-list: five lines, with safeguards.

7. The numbers (10 min). Three to five metrics with a baseline plan. Monthly cost in money, time, and attention. Value per action vs. cost per action.

8. The next two weeks (10 min). A paper walkthrough or a five-case wizard-of-oz run. One named owner. One date on the calendar.

Chapter 16 — Next steps

You now have the whole method: problem before tools, workflow before prompts, humans in charge, boundaries written, evidence kept, costs counted.

If you're a learner: run the build worksheet on a problem from your own life — a club, a team, a side project, a family business. Small and real beats big and imaginary. Bring your map to someone who'll stress-test it.

If you're an educator: every chapter maps to a session, and the exercises are classroom-ready. The curriculum folder in this repository has agendas, rubrics, and facilitator notes; Train-the-Trainer exists if you want to run it yourself.

If you run an organization: start with one workflow that eats staff time. Map it with the people who do the work. Decide honestly whether AI belongs in it — and if it does, you'll now build it in a way your board can defend.

If you own a business: pick the pain that costs you money this month — missed calls, follow-ups, scheduling. Do the cost math in Chapter 11 before spending a dollar on tools.

If you want a partner: Crate South Education runs workshops, cohorts, and pilots, and the first conversation is free. The materials in this guide are the same ones we teach from — no secrets held back, because the method was never the secret. The secret is doing it.

Appendices

Appendix A — Glossary

Applied AI. Using AI tools for specific, checkable jobs inside a real workflow. **Approval gate.** A designed step where a named human must approve before the system acts. **Auditability.** Being able to reconstruct what happened: inputs, decisions, approvals, actions, outputs. **Baseline.** A measurement taken before the system exists, used to judge it later. **Blast radius.** How much damage one wrong action can do. Irreversible + wide = tightest controls. **Boundary.** A written rule about what the system must never do. **Cost per action.** One completed action's share of money, review time, and attention. **Decision point.** A moment in a workflow where a choice changes what happens next. **Evidence.** Records that let others understand and trust the system. **Fallback.** The plan for when an AI step fails or produces unusable output. **Human-in-the-loop.** Designs where humans review, approve, or decide at defined points. **Input.** Anything the system receives: messages, forms, calls, documents. **Kill switch.** The named person, defined trigger, and fast procedure to halt the system. **Knowledge base.** Owned, maintained reference material: prices, policies, schedules, answers. **Never-list.** The written list of what the system must never do, with safeguards. **Operator.** The person who runs the system day to day. **Output.** What people receive: replies, summaries, bookings, reports. **Prototype.** A cheap version built to fail informatively. **Rubber-stamping.** Approving without reviewing — the symptom of an over-long queue. **Safeguard.** The mechanism enforcing a boundary: caps, blocks, triggers, switches. **System canvas.** The one-page planning sheet (Appendix B). **Trigger.** The event that starts a workflow. **Wizard-of-oz.** Prototyping with a human secretly playing the AI's role. **Workflow.** The ordered steps from trigger to outcome, with owners.

Appendix B — The one-page system canvas

Fill every box. Empty boxes are your to-do list. (Printable: `templates/SYSTEM_CANVAS.md`.)

Problem (who / what today / cost)

Trigger (what starts it)

Inputs (sources, formats, permissions)

AI jobs (narrow, checkable, one sentence each)

People (served / operators / touched / accountable)

Workflow (steps, waits, top exceptions)

Knowledge (sources, owners, update rhythm)

Decisions (points, criteria, who decides)

Approval gate (who, how fast, reviewing what)	**Actions** (reversible? logged? limits?)
Outputs (what people receive, quality bar)	**Metrics** (3–5, with baseline plan)
Never-list (5 lines + safeguards)	**Cost** (money + time + attention vs. value)
Stop procedure (who, trigger, how)	**Next step** (one owner, one date)

Appendix C — Responsible-use checklist

Run this before any system touches real people. Every "no" is a task, not a judgment. (Printable: `templates/RESPONSIBLE_USE_CHECKLIST.md`.)

- ☐ The problem statement names who is affected and what it costs them
- ☐ Every affected group has been identified — including people whose data flows through
- ☐ A named human is accountable for the system after launch
- ☐ Inputs are limited to what the purpose requires, with permission trails for personal data
- ☐ Every decision point is marked, with written criteria and a named decider
- ☐ Consequential actions require human approval — fast to review, impossible to skip silently
- ☐ Every action is rated reversible or irreversible; irreversible ones have the strongest controls
- ☐ The never-list is written, safeguards exist, and each safeguard has been tested
- ☐ Failure conditions are listed, with prevent / detect / absorb decisions
- ☐ The stop procedure is written and a tired operator could execute it in five minutes
- ☐ Metrics exist (3–5), a baseline was taken or scheduled, and a review rhythm is set
- ☐ Monthly cost is estimated in money, time, and attention — and compared honestly to value
- ☐ The seven-item documentation set exists and is dated
- ☐ Recipients of outputs have a way to reach a human
- ☐ Opt-outs are honored immediately, every time

Appendix D — Workshop handout version

The whole method on one page.

1. **Problem before tools.** Name who, what happens today, what it costs. Three sentences, no adjectives.
2. **People always.** Served, operators, touched, accountable. Empty seats stop the build.
3. **Workflow before prompts.** Trigger → steps → waits → exceptions, on paper, confirmed by the person who does the work.
4. **AI gets narrow jobs.** Draft, summarize, classify, extract, translate. Checkable in under a minute or it doesn't qualify.
5. **Humans decide and approve.** Decision points have criteria and named deciders; consequential actions wait for a human yes.
6. **Boundaries are a feature.** Five never-lines, each with a safeguard, each safeguard tested.

7. **Evidence over enthusiasm.** 3–5 honest metrics, a baseline, a review rhythm.
8. **Cost is design.** Money + time + attention per action, vs. value per action. If it doesn't pencil, don't build.
9. **Improve on purpose.** Evidence → hypothesis → small change → measure → keep or revert. Log everything.
10. **Stop is designed too.** Named person, defined trigger, five-minute halt.

Crate South Education · From Prompt to System · This material is public; teach with it.