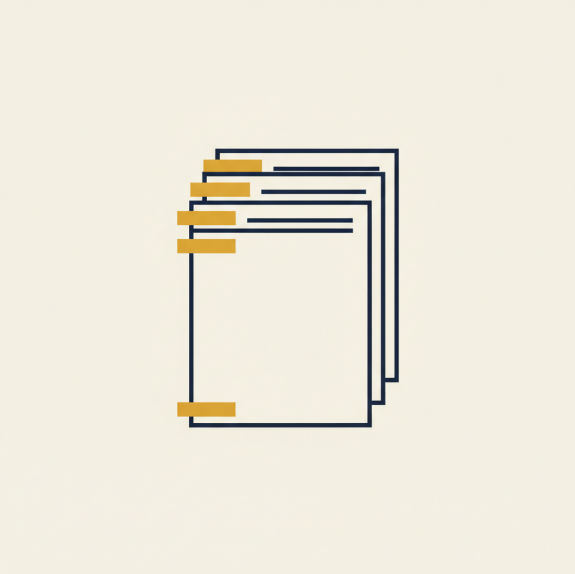
TIER 2 - PROFESSIONAL * V1.0 -- MAY 2026

ROLE-SPECIFIC PROMPT TEMPLATES

How to design prompt libraries that compound across your team -- versioned, governed, organized by role, durable across model upgrades. The piece that sits underneath any real team rollout.



BY Alex Jahn / Agent Logic

v1.0 -- May 2026 **EDITION**

Operators rolling out AI to a team who want a sustainable prompt library, not one-shot prompts that vanish after one use **AUDIENCE**

15-20 minutes **READ TIME**

Free. Forever. **COST**

Prepared by Agent Logic / alexanderjahn79@icloud.com / theaiguywi.com

CONTENTS

What's in here

SECTION 1

The "every employee writes their own prompts"

Eight people, eight versions of the same prompt

Watch what happens at most small teams six months into their AI adoption.

Your front-desk person figured out a decent prompt for drafting customer responses. Your bookkeeper figured out a different one for summarizing vendor emails. Your project manager has yet another for status reports. None of them have ever shown each other their prompts. Each one rebuilt the wheel. Each one rebuilt it slightly differently. Each one is mid-quality at best because they're fighting the same battle alone.

Six months in, your team has eight people, eight different ways of writing the same kind of prompt, eight different output qualities, and zero institutional memory. Hire someone new and they start from scratch. Lose someone and their prompts walk out the door. Upgrade your AI subscription and nobody knows which prompts are still optimal.

This is the predictable failure mode of "we use AI as a team." Not a tooling failure. Not a training failure. A *library* fail

This module is how to build the library that fixes it.

What this module is and isn't

It's Scrap
templates as a starting point -- front desk, sales, ops, trades. Use those.

This module is the layer underneath. *How to B*
collected prompt work. Library structure, naming conventions, versioning, governance, the design properties that make a prompt actually reusable. The unsexy infrastructure that turns prompts from disposable artifacts into institutional capital.

By the end you'll have:

- A clear picture of what a prompt library is (and isn't).
- The four design properties of a reusable prompt.

- A walkthrough for building your first three real templates from scratch.
- The cross-role patterns every library needs.
- A working approach to versioning + governance -- plain English, no devops jargon.

> ***A prompt used once is a tool. A prompt used 200 times is institutional capital. The difference isn't the prompt. It's whether your team can find it the second time.***

SECTION 2

What a prompt library actually is

Structure first, content second

A prompt library, at its dumbest, is a folder. A folder organized by role, with one document per template. Each document contains four things:

> **The four things every template document contains:**

1. **Name** -- Short, descriptive, role-prefixed. *"FrontDes* *"my emai* *prompt v3 final FINAL."*
2. **Purpose** -- One sentence. What this template is for. When to use it. When NOT to use it.
3. **Prompt text** -- The actual prompt, copy-paste ready. With placeholders for the parts the user fills in.
4. **Example output** -- One real example of what the model produced when this prompt was run. Anchors expectations. Catches model drift over time.

That's it. Four things. Anyone on your team can read a template doc in 30 seconds and decide whether it's the right tool for their current task.

Where the library lives

The library doesn't have to live in a fancy tool. Three perfectly fine options:

- **A shared folder** (Google Drive, Dropbox, Box) with subfolders by role and one Markdown or text file per template. Free, instantly searchable, version history if your platform tracks it.
- **A shared Notion / Coda / similar workspace** with a database of templates filterable by role, tag, status. Slightly heavier; better if your team already lives in one of these.
- **A simple internal wiki page** organized by role, one section per template, with table of contents. Works if your team doesn't have any of the above and you want zero new tool overhead.

Pick the option your team will actually open. The fanciest tool is the one nobody opens. The simplest tool everyone uses beats it every time.

Folder structure that holds up

``` prompt-library/ ??? shared/ (templates that work across departments) ? ??? summarize-meeting.md ? ??? polite-decline-email.md ? ??? review-doc-for-tone.md ??? customer-service/ ??? sales/ ??? operations/ ??? finance/ ??? leadership/ ```

Subfolders by role; one file per template; a `shared/` section for cross-role workhorses. That's the default structure. You'll evolve it as your library grows.

The mistake most teams make: starting with a fancy taxonomy nobody can remember. Start dumb. Five role folders, one shared folder, one file per template. Reorganize later if you need to.

SECTION 3

The four design properties of a reusable prompt

A template that gets reused 200 times has four properties that one-shot prompts don't. Internalize them and you'll catch design problems before they ship to your team.

Property 1 -- Composability

A composable template has clearly marked **placehold** instead of being one frozen block of text.

Bad (frozen):

> *"Reply to John Schmidt about the kitchen remodel quote we sent on April 14, in a friendly tone, asking if he has any questions before deciding."*

Good (composable):
```

> *"Reply to {{customer_name}} about the {{job_type}} quote we sent on {{quote_date}}, in a {{tone}} tone, asking if they have any questions before deciding."*

The composable version works for every customer, every job, every date. The frozen version is a one-shot. Make every template composable from day one.

Property 2 -- Specificity

A specific template is constrained enough that the output is consistent. The model has freedom in *how* to fill the gaps, but no freedom on *what* the c

Generic template -- produces 7 different outputs across 7 uses:

> *"{{Customer_name}} asked about {{topic}}. Write a reply."*

Specific template -- produces consistently-shaped output every time:

> *"Write a 4-sentence reply to {{customer_name}} about {{topic}}. First sentence: acknowledge their question. Second sentence: direct answer. Third sentence: any caveat or follow-up question. Fourth sentence: warm closing. No marketing language. No 'Dear customer.'"*

The specific version is what reuse looks like. Every time anyone on your team uses it, the output has the same shape.

Property 3 -- Robustness

A robust template still works after a model upgrade.

Models change. ChatGPT-4 -> ChatGPT-5. Claude Sonnet -> Claude Sonnet 5. Output behavior shifts. A template tuned to one model's quirks (e.g., "*put the word 'TLDR:' at the start of your reply because GPT-3.5 ignores formatting otherwise*") will break on the next model.

Robust templates avoid model-specific hacks. They state requirements directly: *"Reply wit* *one-sentence summary first, then the body."* Same intent, future-proof phrasing.

When you do need a model-specific tweak, document it in the Purpose line: *"Tuned fo* *Sonnet 4.6. Verify output on next model upgrade."*

Property 4 -- Documented

Six months from now, someone new joins your team. They open your library and see 47 templates. *Why does the customer-service folder have one called "Apology - long form" and another called "Apology - short form"? When do I use which?*

If your one-line Purpose statement answers that, the new person uses the right template. If it doesn't, they pick wrong half the time.

Every template doc gets one sentence on what it's for and one sentence on when NOT to use it. That's the documentation budget. Skip it and your library becomes a graveyard.

> ***Composable, specific, robust, documented. Miss any one of these and your "library" is just a graveyard of last week's prompts.***

SECTION 4

Building your first three templates -- a walkthrou

Theory's enough. Here's how to actually make this happen at your team this week.

Step 1 -- Pick the role most likely to use AI heavily

Look at your team. Which role does the most repeated, AI-eligible work? Usually it's customer service / front desk, sometimes sales, sometimes ops. Whichever role has the most reused prompts already in heads, pick that one.

You're going to build three templates for that role first. Three is the minimum useful library -- fewer feels like nothing, more is too much to maintain on day one.

Step 2 -- Pick the three tasks that role does most often

Sit with that person for 15 minutes. Ask: "*What are the three things you do every day that an AI prompt could help with?*" They'll tell you. Don't argue with their answer -- they know better than you.

Typical answers from front-desk:

- Drafting first-reply emails to inbound customer inquiries.
- Summarizing yesterday's voicemails into a status doc.
- Polite-decline replies to ad-hoc vendor outreach.

Typical answers from sales:

- Drafting proposal cover emails.
- Three-day-no-reply follow-ups.
- Responding to "*can you do this for less?*" objections.

Whatever the three are, that's your first three templates.

Step 3 -- Build template one. Battle-test it.

Build the template per the four properties (composable, specific, robust, documented). Hand it to the person who does the work. Ask them to use it for one week.

The deal: every time they use it, they note what they had to change in the output. Not the prompt -- the output. Did they have to delete a sentence? Add one? Reword the closing? Those notes are your refinement queue.

Friday afternoon: go over the notes together. Update the template based on what consistently needed changing. Mark version 2.

That's how a real template gets built. Not in a meeting, not in a strategy doc -- in a week of actual use, with refinement based on real friction.

Step 4 -- Repeat for templates two and three

Same loop. By the end of three weeks, that role has three working templates that are battle-tested in real use.

The whole library starts there. Three templates × five roles = 15 templates after about four months of disciplined building. That's a working library for a small business. You're not building enterprise-grade. You're building useful.

3

Templates per role to start.

Don't try to launch a library with 30 templates that nobody battle-tested. Three real ones beats thirty theoretical ones every time.

SECTION 5

Cross-role patterns

Some prompts apply across every role at your company. They're the highest-reuse-rate templates in any library. Build them once, file them under a shared section, watch them get used by everyone.

The big four:

> **The four cross-role workhorses every library should have:**

1. **Summarize this meeting.** *"Summari*
   made, action items with owners, open questions. Tone: neutral, no editorializing."
2. **Draft a polite reply declining a request.** *"Draft a 3-*
   message below. Stay warm but clearly say no. Don't over-explain. Don't apologize more than once."
3. **Review a customer-facing doc for tone.** *"Review t*
   Flag (a) any marketing language that should be plainer, (b) any ambiguity a customer might misread, (c) any place we sound apologetic when we shouldn't. Suggest specific edits, don't rewrite the whole thing."
4. **Compress a long document into talking points.** *"Compres*
   talking points for a 10-minute meeting. Lead with the most surprising or load-bearing fact. Skip background everyone already knows."

These four show up in every department. They're worth their weight in gold and they're easy to build. If your library has nothing else, it should have these four.

Why the shared section beats per-role duplicates

Without a shared section, each role builds their own version of "summarize this meeting" and you end up with 5 nearly-identical templates, each with slightly different output behavior. Maintenance becomes a nightmare and quality drifts.

With a shared section, one canonical version exists. It gets refined by everyone who uses it. It improves over time across the whole company.

The shared section is the highest-leverage 30 minutes you'll spend on your library.

SECTION 6

Versioning and governance

Once your library has 10+ templates, you have a maintenance question. Without governance, it rots. With it, it compounds.

When to update a template

Three triggers:

- **Model upgrade.** New version of your default model ships. Test your top 5 templates on the new model. Most will work fine. Some will need tweaks. Document them.
- **Output drift.** People keep complaining the output isn't quite right. Look at the recent uses. Update the template. Bump the version number.
- **New edge case.** A scenario the original template didn't anticipate keeps coming up. Add a clause or build a variant template. Document the difference in Purpose lines.

Naming versions

Don't get fancy. *v1, v2, v3.* `customer-service/first-reply-email-v2.md`. Keep prior versions in an archive folder for two months in case the new version misfires and you need to roll back.

Who owns the library

One person. Pick one. They don't have to be the AI expert -- they just have to be the person who notices when something's broken and either fixes it or escalates. At a 10-person company this is usually a 30-minute-a-week job. At a 50-person company maybe an hour a week.

Without an owner, the library becomes everybody's problem, which means it's nobody's problem. Pick one person.

How to deprecate

Templates go stale. New ones replace them. Old ones get used out of habit and produce wrong output.

Deprecation flow: mark the old template's filename with a `DEPRECATED-` prefix. Move it to an `archive/` subfolder. Add a one-liner at the top: *"DEPREC [new-template-link]. Reason: [reason]."* Keep it for two months in case anyone hits it from an old reference. Then delete.

That's library hygiene. Not glamorous. Saves you from drift.

SECTION 7

The team rollout connection

This module covered the *library*. Th
versioned, governed. That's the asset.

The *rollout* is t
Showing-and-telling. Onboarding. Resistance from the people who don't want to learn yet another thing.

That's a different module -- it's **Strap In:**
this module first and you're nodding along thinking *"I have the*
how to actually launch it," -- Strap In is the next read. Same curriculum, deeper into the people side.

The relationship in one sentence: **the librar**

SECTION 8

Where to go from here

You're now nine modules into the curriculum. Tier 1 (Personal - 6) is complete; this is module 3 of 6 in Tier 2 (Professional). Three Tier 2 modules left:

- **Multi-step task chains** -- when one prompt isn't enough; breaking work into reliable sequences.
- **Picking the right model** -- ChatGPT vs Claude vs Gemini vs open-source; cost-per-task economics from a real operator.
- **Privacy and what not to paste** -- the workplace version of Tier 1's privacy module; data classifications, consumer vs enterprise tiers, the 1-page policy.

Get the next module the day it drops: theaiguywi.com/training

One email per release. No drip. No spam. Opt out anytime.

If you want me to come build your prompt library *with* you -
high-reuse tasks, build the first 15 templates, install the governance, and hand it off -- that's the consulting offer. Same install I run on my own carpentry business. Done with you, not lectured at you.

Reach out: alexanderjahn79@icloud.com

A short call. Honest scope. We figure out together if it's a fit.

Closing -- the lock-in line

The single number that matters most about a prompt library:

200

Times a good template gets reused per year, per role.

Build it once, well -- composable, specific, robust, documented -- and your team gets the multiplier 200 times. Build it badly or not at all and your team rebuilds the wheel every Tuesday for the rest of time. The library is institutional capital. The rest is tactics.

You have the structure. The next module is what to do when one prompt isn't enough.

Agent Logic -- *theaiguyw*

Fond du Lac, WI. This is module 3 of 6 in Tier 2 (Professional).