DESIGNING FOR AGENTIC AI: A SKILLS FRAMEWORK

How to Think About “Skillification” from a Design Perspective

---

INTRODUCTION

As organizations adopt agentic AI systems, designers face a new challenge: how do you break down user needs into capabilities that autonomous agents can execute? This isn’t a technical architecture problem — it’s a design thinking problem.

This framework helps you evaluate what should become an agent “skill,” how to prioritize skills for development, and how to collaborate effectively with product and engineering teams on agentic systems.

---

1. WHAT IS A “SKILL” IN AGENTIC AI?

A **skill** is a discrete capability that an agent can invoke to accomplish a specific sub-task within a larger goal. Think of it like a tool in a toolbox — the agent reaches for the right one based on what the user needs in that moment.

From a design standpoint, a skill has three characteristics:

1. **A clear input** — What information does the agent need to execute this?
2. **A defined output** — What does the agent produce or accomplish?
3. **A single responsibility** — It does one thing well, not five things poorly.

Example:
In a customer self-service experience, “check order status” is a skill. It takes an order ID (input), retrieves the current status and timeline (output), and does that one job (single responsibility). It doesn’t also try to process returns or recommend products — those are separate skills.

---

2. THE DECISION FRAMEWORK: IS THIS A SKILL?

When you're looking at current functionality and trying to figure out if it should be "skillified," ask these questions:

Question 1: Can this be invoked independently?

Test: Could an agent call this capability on its own without needing to execute five other things first?

- ✅ **Yes = Skill candidate**
  Example: "Retrieve product availability for a given SKU"
- ❌ **No = Probably part of a larger workflow**
  Example: "Handle the entire checkout flow from cart to confirmation" — this is a *workflow* made up of multiple skills, not a single skill.

Question 2: Does it have a clear success state?

Test: Can you say definitively when this task is "done"?

- ✅ **Yes = Skill candidate**
  Example: "Find store locations within 10 miles of this ZIP code" — done when the list is returned.
- ❌ **No = Needs to be broken down further**
  Example: "Help customer save money" — too vague. What does "help" mean? This is a *goal*, not a skill. The skills underneath might be: compare pricing, identify promotions, suggest alternatives, apply discount codes, etc.

Question 3: Is it reusable across different user journeys?

Test: Could multiple different experiences benefit from this same capability?

- ✅ **Yes = Strong skill candidate (high leverage)**
  Example: "Verify customer account status" — needed for purchases, support inquiries, account changes, loyalty programs, etc.
- ❌ **No = Might be too niche, or might be part of a larger skill**
  Example: "Display the specific error message when a gift card balance is depleted" — this is probably just UI logic inside a larger "check gift card balance" skill, not its own skill.

Question 4: Does it require access to a specific data source or system?

Test: Is there a discrete integration or data lookup happening here?

- ✅ **Yes = Often a good skill boundary**

Example: “Query inventory management system for real-time stock levels” — clean integration point.

- ❌ **No = Might just be logic, not a skill**
  Example: “Format a currency amount with proper symbols” — that’s just code, not a skill an agent needs.

---

3. SKILL GRANULARITY: ATOMIC VS. COMPOSITE

Not all skills are created equal in terms of size and complexity. You’ll encounter two types:

Atomic Skills (Single-Step)

These do **one specific thing** and nothing more. They’re the building blocks.

Examples:

- Look up customer’s current account balance
- Retrieve list of available shipping options for a destination
- Check if a product requires special handling
- Find nearest service location to a given address

Design implication: Atomic skills are easy to test, easy to explain, and very reusable. But they don’t solve a user’s problem on their own — they need to be *composed* into something meaningful.

Composite Skills (Multi-Step)

These **orchestrate multiple atomic skills** to accomplish a higher-order task.

Examples:

- “Explain why my bill changed this month” (composite of: retrieve recent transactions, pull account details, compare to prior period, identify deltas, format explanation)
- “Find the best shipping option for this order” (composite of: check product dimensions/weight, query shipping rates across carriers, apply customer preferences, calculate delivery windows, rank by cost and speed)
- “Help me choose the right service plan” (composite of: retrieve all available plans, analyze customer’s historical usage, project future costs under each plan, compare side-by-side, recommend)

Design implication: Composite skills are what the user actually experiences as "helpful." But under the hood, they're calling a bunch of atomic skills in sequence. When you're designing, you need to think about both layers: the atomic capabilities AND the composite experiences they enable.

---

4. HOW SKILLS COMPOSE INTO END-TO-END EXPERIENCES

Let's walk through a concrete example to make this real.

User Goal: "I need to know the total cost including shipping for this order."

This feels like one thing from the user's perspective. But as a designer, you'd break it down like this:

The end-to-end experience is powered by these skills:

1. **Verify customer account status** (atomic)
   Input: Customer ID
   Output: Active/inactive status, membership tier
2. **Calculate order subtotal** (atomic)
   Input: Cart contents, current pricing
   Output: Subtotal before shipping and tax
3. **Retrieve customer's shipping address** (atomic)
   Input: Customer ID
   Output: Default or selected shipping address
4. **Query available shipping options** (atomic)
   Input: Order details, destination address
   Output: List of carriers with rates and delivery windows
5. **Calculate sales tax** (atomic)
   Input: Subtotal, shipping cost, destination address
   Output: Tax amount based on jurisdiction
6. **Apply loyalty discounts** (atomic)
   Input: Customer ID, membership tier, order total
   Output: Discount amount if applicable
7. **Present total cost breakdown** (composite — wraps all of the above)
   Input: User's question ("How much will this cost?")
   Output: "Your order total is $127.43: $98 for items, $15.99 shipping, $13.44 tax. You're

saving $12 with your membership discount. Want me to proceed with checkout?"

What you're designing:

- The **atomic skills** (1-6) are the capabilities the agent needs to exist. These are what product/engineering builds.
- The **composite skill** (#7) is the *experience* the user has. This is what you design the conversational flow and UI around.
- The agent decides *when* to invoke which skill based on context. Your job as a designer is to make sure the right skills exist and that they compose in a way that feels seamless to the user.

---

5. PRIORITIZATION: WHAT MAKES ONE SKILL MORE VALUABLE THAN ANOTHER?

Not everything needs to be a skill right away. Here's how to think about prioritization when you're working with product teams:

High-Priority Skill Candidates:

1. High frequency, high friction
If users are asking this question constantly and the current experience is frustrating, skillify it.
Example: "Where is my order?" — happens all the time, current answer requires hunting through multiple systems.

2. Reusable across multiple journeys
If this capability unlocks value in 5 different places, it's high leverage.
Example: "Verify account status" — needed everywhere. Build it once, use it everywhere.

3. Requires real-time data
If the answer changes frequently and stale info is worse than no info, this needs to be a skill the agent can call on-demand.
Example: Inventory levels, order status, service availability.

4. Regulatory or compliance-critical
If getting this wrong has legal/compliance implications, it needs to be a governed skill with clear audit trails.
Example: Identity verification, financial calculations, age-restricted product checks.

5. Enables a "wow" moment

If this unlocks something the user couldn't do before (or could only do with huge effort), it's a differentiator.
Example: "Find comparable products at lower prices and show me quality comparisons" — most users have no idea this is even possible.

Lower-Priority (Defer for Now):

1. Edge cases with low volume
If only 2% of users ever need this, it's probably not a skill yet — handle it as an escalation to a human.
Example: "Explain the warranty terms when I have three overlapping protection plans."

2. One-time or rare events
If this only happens once per customer per year, the ROI on skillifying it is low.
Example: "Help me set up autopay for my account" — important, but infrequent.

3. Highly variable, low-structure tasks
If there's no clear pattern or the answer is "it depends" more than "here's the answer," it's not ready to be a skill.
Example: "Help me decide which gift to buy for my friend" — too much nuance, needs human judgment (or at least a very sophisticated composite skill you're not ready to build yet).

---

6. HOW TO COLLABORATE WITH PRODUCT & ENGINEERING ON SKILLS

Here's how to think about your role in these conversations:

What Product/Engineering Will Likely Focus On:

- Technical feasibility: "Can we build this skill given our current integrations?"
- Data availability: "Do we have access to the data this skill needs?"
- Performance: "How fast does this need to respond?"
- Scalability: "How many calls per day will this skill handle?"

What You Should Focus On:

- User value: "Does this skill unlock a moment that matters to users?"
- Experience continuity: "How does this skill fit into the larger journey?"

- Conversational design: “How does the agent explain what this skill is doing / what it found?”
- Error states: “What happens when this skill fails or returns no results?”
- Transparency: “Does the user need to know this skill was invoked, or does it happen invisibly?”

The Questions You Should Be Ready to Ask:

- “If we build this as a skill, what does the agent do with the output? How does it get presented to the user?”
- “What’s the fallback if this skill errors out? Does the agent have a graceful way to recover?”
- “Is this skill something the agent invokes automatically, or does the user need to explicitly ask for it?”
- “How do we explain to the user what just happened when this skill runs?” (Especially important for transparency in regulated environments.)

---

7. A SIMPLE TEMPLATE FOR EVALUATING A POTENTIAL SKILL

When you encounter a piece of functionality and you’re wondering “should this be a skill?”, run it through this:

Question	Your Answer	Implication
What user goal does this support?	[e.g., “Customer wants to know product availability”]	If you can’t articulate the user goal, it’s probably not a skill yet — it’s just a capability looking for a purpose.
What is the single thing this does?	[e.g., “Retrieves current inventory levels for a given product and location”]	If you can’t describe it in one sentence, it’s either too complex (break it down) or too vague (sharpen it).
What information does it need as input?	[e.g., “Product SKU, store location or ZIP code”]	If the input list is huge or unclear, the skill boundary isn’t well-defined.

What does it produce as output?	[e.g., “In-stock quantity or estimated restock date”]	If the output varies wildly or is unpredictable, the skill may not be mature enough yet.
Can it be reused in other journeys?	[e.g., “Yes — inventory check needed for purchase, reservation, store pickup, customer service”]	High reusability = high priority.
Does it require real-time data?	[e.g., “Yes — inventory changes constantly”]	Real-time dependencies make this a stronger skill candidate.
What happens if it fails?	[e.g., “Agent says ‘I can’t check availability right now, but I can connect you to a store associate’”]	If there’s no good fallback, the skill isn’t production-ready.

---

8. FINAL THOUGHT: SKILLS ARE LEGO BLOCKS, EXPERIENCES ARE WHAT YOU BUILD

The way to think about this:

- **Skills are the Lego blocks.** Atomic, reusable, well-defined capabilities. Product and engineering build these.
- **Experiences are what you build with them.** The composite flows, the conversational arcs, the moments that matter to users. You design these.

Your job isn’t to define every skill. Your job is to:

1. Understand what skills exist (or could exist)
2. Identify which ones are missing when you’re designing an experience
3. Recommend to product/engineering: “We need this skill to make this experience work”
4. Design how those skills get invoked, explained, and recovered from when they fail

The better you get at thinking in terms of “what skills does this experience need?” the more effective you’ll be in product conversations. You’ll be able to say things like:

- “This experience requires three skills: account verification, inventory lookup, and price calculation. We have the first two, but the third one doesn’t exist yet. Can we build it, or do we design around the gap?”

- “You’re proposing a skill that does five things. I think we should break it into two skills so they’re more reusable.”
- “This skill is technically feasible, but from a user perspective, it doesn’t map to anything they’d actually ask for. Let’s shelf it and focus on the ones that unlock real user value.”

CONCLUSION: SKILLIFICATION AS A DESIGN DISCIPLINE

What you’re really building here is a **design language for agentic systems** — a shared vocabulary and evaluation framework that lets design, product, and engineering have productive conversations about what to build and why.

The organizations that figure this out early (how to think about skillification as a design discipline, not just an engineering task) will move faster than those still treating it as a black-box technical problem.

The AI technology isn’t what you need to worry about. It’s how you structure the problem. And that structuring is actually quite straightforward when you approach it with design thinking principles:

- What’s the user trying to do?
- What capability do we need to help them do it?
- Is that capability reusable?
- How does it fail gracefully?
- What’s the priority?

These are questions designers already know how to answer. The “agentic AI” part is just the delivery mechanism.

This framework can be adapted for any industry or domain where agentic AI systems are being designed and deployed.