RED CROSS AI SUBGROUP · STUDY GUIDE

What Salesforce Learned From 20,000 Agents

A design pattern for blending deterministic rules with generative AI — and the operational discipline that makes agents survive real users.

Source: ByteByteGo — interview with John Kucera, CPO of Agentforce

Prepared for Jeff Franzen · June 11, 2026 · Companion to the `agent-building-lessons` skill & Ask Clara

> *An excellent design-pattern overview of how Salesforce itself builds agent workflows with a blend of deterministic rules and genAI capabilities, and the rise of the agent manager role in operations. This design pattern could apply to any stack. A highly recommended read to inspire our thinking about how to approach this type of work.*
>
> — Red Cross AI subgroup lead, sharing the article

01 The one big idea

The demo is the easy part. The real work starts after you turn it on.

Anyone can cook one good meal for a friend and feel like a chef. Running a restaurant — strangers ordering things you didn't plan for, every night — is the actual job. Most AI agents die because the builder cooks the one good meal, shows the boss, gets a thumbs-up, and stops. Then real users arrive with edge cases the demo never saw. **Treat "it works in the demo" as the starting line, not the finish line.**

The credibility behind this: Salesforce has watched **20,000 companies** run agents in production; their own support agent has handled **3 million+ conversations**.

02 The 4-layer stack (the mental map)

Every agent lives in four stacked layers, with security wrapping all of them:

LAYER	WHAT IT IS
Engagement	Where a human talks to the agent — chat, Slack, phone. The front door.
Agent	The brain. The AI reasons and decides. Where you build, monitor, orchestrate.
System of work	The real apps where work gets done — resolve a case, process a return. The hands.
Context	The data and background that ground answers in *your* reality, not generic knowledge.

Wrapping all four: the **trust layer** — security and guardrails, supporting multiple AI providers.

---

03 The 90% flip

Traditional software

~90% of the work is **before** launch — design, build, test. After launch you're in maintenance.

AI agents

~90% of the work is **after** launch — managing and improving as real questions arrive.

Kucera's words: *"In the typical software world, 90% of the work is getting to go live. Whereas in the typical AI agent, 90% of the work is after you go live to manage and improve the agent."* The failure mode: teams use the old playbook and assume they're done at launch. The demo handles *typical* questions — but typical questions are the minority of what real users ask.

---

04 Deterministic + genAI — the design philosophy

This is the boss's headline point and the intellectual core of the article.

- **The LLM is probabilistic** — predicts the next words. Flexible with messy human language, but inconsistent (same question, different steps).
- **Code is deterministic** — same input, same output. Reliable but rigid.

The pattern

The future isn't one or the other — it's **both, layered**. Deterministic rules set the guardrails and run the predictable parts; the probabilistic AI floats on top for the genuinely fuzzy parts. Salesforce built **Agent Script** (encode predictable flows as code) and **Hybrid Reasoning** (mix fixed logic with AI) precisely for this. *This applies to any stack — including ours.*

---

05 Pre-launch — get the foundation right

1. Start small ("don't boil the ocean")

Pick one use case that's **both high-value and achievable**. Models change fast (overbuild today, rebuild in six months), and your team has to learn a brand-new skill — reading transcripts, diagnosing wrong calls, updating instructions/tools/data. That learning is faster and safer on something small.

2. Tie the agent to a KPI

- **Containment rate** — % of cases fully resolved with no human follow-up.
- **Agentic Work Units (AWUs)** — a unit for *meaningful work actually completed*. The point: activity ≠ value. A chatty agent that resolves nothing scores low.

The KPI also tells you what to fix first: a tone annoyance loses to a logic error that tanks containment.

---

06 The trust layer — guard both directions

The agent sits between your users and your data with an AI in the middle. Data flows **in** (your question + your data) and **out** (its answer + actions). Guard each side — neither alone is enough.

Data going IN

GUARD	WHAT IT DOES
Secure retrieval	Don't give raw database access; route through a layer that returns only what this agent may see.

Zero data retention	Provider contract: won't store your prompts/answers, won't train on them. **The key control for humanitarian data.**
Trusted boundary	Run the model inside your own walls so data never crosses the public internet (e.g. Claude hosted inside Salesforce's boundary).
Data masking — a trap	Hiding fields strips the context the AI needs to reason. Keep it **off by default**; use only where the hidden field isn't needed.

Answer coming OUT

GUARD	WHAT IT DOES
Tool validation	Block invented actions — the AI hallucinates *actions*, not just words.
Grounding checks	Verify the answer came from approved sources, not the AI's general memory.
Content filtering	Screen toxic or harmful output before anyone sees it.

07 Post-launch — the feedback loop

Agent failures are fuzzy, not pass/fail. Triage every one into four buckets, each with a different fix:

FAILURE	WHERE TO FIX IT
Tone / brand voice	The system prompt / instructions
Logic (wrong tool, too many steps)	Tool config; if it recurs, make it **code**
Wrong answer from a bad source	Fix the **source data**, not the agent (Salesforce grounds on 135k help articles, constantly finds stale ones)
Question it was never built for	Expand scope or hand cleanly to a human — and **log the gap**

The meta-lesson: the *speed* of this loop is the gate to scaling. Fast loop → trust in your numbers → approval to expand. Slow loop → stuck in pilot forever.

---

08 The three anti-patterns

1. AI reasoning where code is better

"Where's my order?" is a lookup, not a thought. Routing it through AI reasoning adds round-trips, latency, and error chances. This birthed **Agent Script**. Rule: **if you can draw it as a flowchart, it's code, not a prompt.**

2. Prompting harder instead of encoding policy

Teams write `NEVER do X!!!` in bold caps. It doesn't work — the AI doesn't respond to emphasis like humans. Hard rules ("we don't operate in Hawaii") become **conditionals in code**: if state = Hawaii, return this exact response. No AI judgment.

3. Poor context engineering

Dumping full data hurts both speed and accuracy — the needle gets buried. One `get_orders` call returned ~100,000 tokens; trimmed to ~2,000 (just ID, status, delivery date, tracking) it was **faster AND more accurate**. Less context wins.

---

09 The "agent manager" role

"90% of the work is after launch" is **a job, not a phase.** The **agent manager** owns the live agent in operations:

- reviews transcripts regularly,
- triages every failure into the four buckets,
- owns the KPI and keeps the fix-it loop fast,
- decides when a recurring flow should become deterministic code.

For Red Cross this is an **operations role distinct from the builders.** The subgroup will need to name *who manages* each agent in production — not just who ships it.

10 Where it's heading

- **Multi-agent orchestration** — a parent agent coordinating specialized sub-agents (Salesforce goes three levels deep). Decompose into small specialists rather than one giant brain.
- **Agents beyond the chat window** — multi-day tasks, background agents with no UI, agents across phone/email/web/Slack. Keep the brain independent of the delivery channel.
- **The pace of change** — coding agents went from basic to powerful in months. Best practice today may be stale in six. Build on durable disciplines, not this month's trick.

11 Applied to Ask Clara

Ask Clara (clara.jbf.com) is a citation-first RAG policy assistant for ARC disaster response — the textbook case for this article.

PASSED

Lean context

`TOP_K=3`, ~1,800 chars/source. ~3 tight chunks, not a token dump. (Anti-pattern #3.)

PASSED

Code-built citations

Prompt forbids inline citations; sources render as cards from chunk metadata. Deterministic, not prompted. (Anti-pattern #2.)

GAP

Grounding is prompt-only

Anti-hallucination relies on a worded instruction. Add a *code* grounding check

GAP

Reranker off by default

`ENABLE_RERANKER=0`. The cross-encoder is the main accuracy lever — A/B it on.

(retrieval-score threshold → withhold/flag).

GAP

No KPI / coverage log

`batch_test.py` is the bones. Define one metric; log coverage gaps so the corpus grows where users ask.

GAP

Confirm zero retention

Clara ships ARC policy text to the API. Verify a zero-data-retention agreement is in place.

Sharpest takeaway for Clara: when it's wrong, suspect the *policy document and retrieved chunks* before touching the prompt. Corpus hygiene is the agent's accuracy.

12 The pocket version

1. Demo working ≠ done. The job is 90% after launch.
2. Start with one small, high-value use case.
3. Pick one number that proves it works (containment / AWUs).
4. If it's a flowchart, it's code — not a prompt.
5. Encode rules in code; don't yell at the AI in the prompt.
6. Feed it only what it needs — lean context is faster AND more accurate.
7. Build a fast fix-it loop; its speed gates your ability to scale.
8. Guard both sides: data going in, answers coming out.
9. Name an agent manager to run it in production.

Study guide compiled from What Salesforce Learned From 20,000 Agents. Companions: vault note `projects/red-cross/enterprise-ai-agents-salesforce-20k.md` · skill `agent-building-lessons` · Clara mapping `ask-clara/docs/notes/20k-agent-lessons-applied.md`.