Step-Based Flows: How ASAPP solved the reliability problem for enterprise AI agents

Large language models are remarkably good at conversation. They interpret nuance, adapt tone, handle ambiguity, and respond with a fluency that can feel human. For customer experience leaders, this capability represents a transformation: AI agents that don't sound like robots, that can navigate the messy reality of customer conversations, and that can resolve issues without forcing customers through rigid decision trees.

But there's a problem that every enterprise encounters the moment they move beyond a demo: these same models make mistakes. They skip steps. They hallucinate policy details. They call the wrong API. They forget to verify account ownership before issuing a refund. They work perfectly in testing and then fail unpredictably in production—not frequently enough to be obviously broken, but often enough to erode trust.

This is the core tension. The very quality that makes LLMs compelling for customer interactions—their flexibility and expressiveness—is precisely what makes them unreliable for executing certain business processes. An LLM that can improvise a natural response can also improvise a wrong one.

For enterprises with regulatory obligations, complex workflows, and millions of customer interactions, "usually correct" isn't good enough. The question isn't whether generative AI agents can have great conversations; it's whether they can be trusted to follow the rules while doing so.

CXP Step-Based Flows is ASAPP's answer: an architecture that preserves the natural, human-like quality of LLM-powered interactions while guaranteeing that business logic executes exactly as designed. **You don't have to choose between a great customer experience and operational confidence. You get both.**

Why prompts alone aren't enough

Before we describe how Step-Based Flows works, it's worth understanding why the most common approach to building AI agents, configuring them through a single set of instructions, runs into trouble at enterprise scale.

Most AI agent platforms work by giving the LLM a large prompt: a block of text containing the agent's persona, policies, available tools, and business rules. The LLM reads these instructions and does its best to follow them while carrying on a conversation. For simple use cases, this works surprisingly well.

As complexity grows, the cracks appear:

- **Instructions interfere with each other.** When a prompt contains hundreds of rules spanning multiple workflows, changing one instruction can unpredictably affect behavior elsewhere. Rewording a refund policy might alter how the agent handles authentication. Not because they're related, but because the model's interpretation of the entire instruction set shifts. Teams end up in a cycle of fixing one thing and breaking another.

- **The agent skips steps.** An LLM juggling dozens of tools and hundreds of instructions will occasionally skip a validation, call the wrong API, or forget to collect a required piece of information before proceeding. These failures are intermittent and hard to reproduce, which makes them especially difficult to diagnose and fix.

- **Everything isn't visible.** The conversation flow exists only as prose in a prompt. It can't be visualized, statically analyzed, or inspected by QA before deployment. When something goes wrong, there's no way to trace the problem to a specific point in the process.

- **Testing is inadequate.** Without structured steps, there's no way to unit test individual parts of a workflow. Teams resort to end-to-end conversation testing, which is slow, incomplete, and doesn't isolate where failures occur.

None of this means prompts are useless—far from it. Natural language instructions are the right interface for guiding how an AI agent communicates. The issue is asking prompts to do double duty: controlling both *how the agent talks* and *what the agent does*. Step-Based Flows separates these concerns.

How Step-Based Flows works

CXP Step-Based Flows is built on a core architectural insight: **the LLM should handle the conversation, and a deterministic state machine should handle the business logic.** Neither is responsible for the other's job.

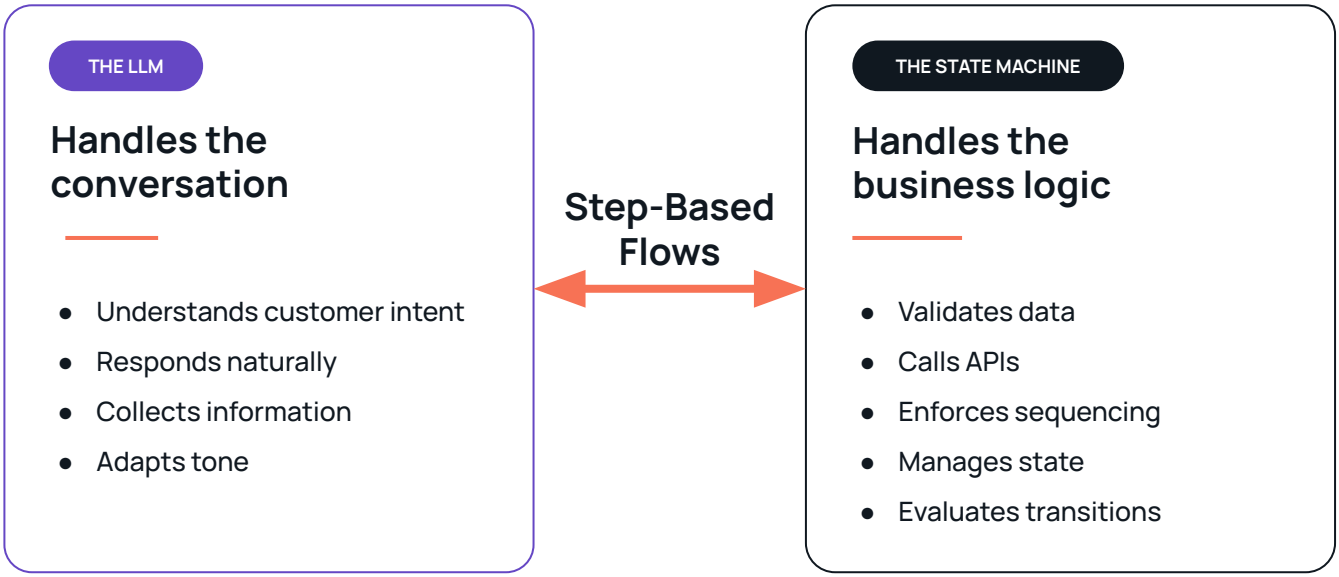


Figure 1. The LLM and the state machine work together at each turn.

Workflows as step-by-step graphs

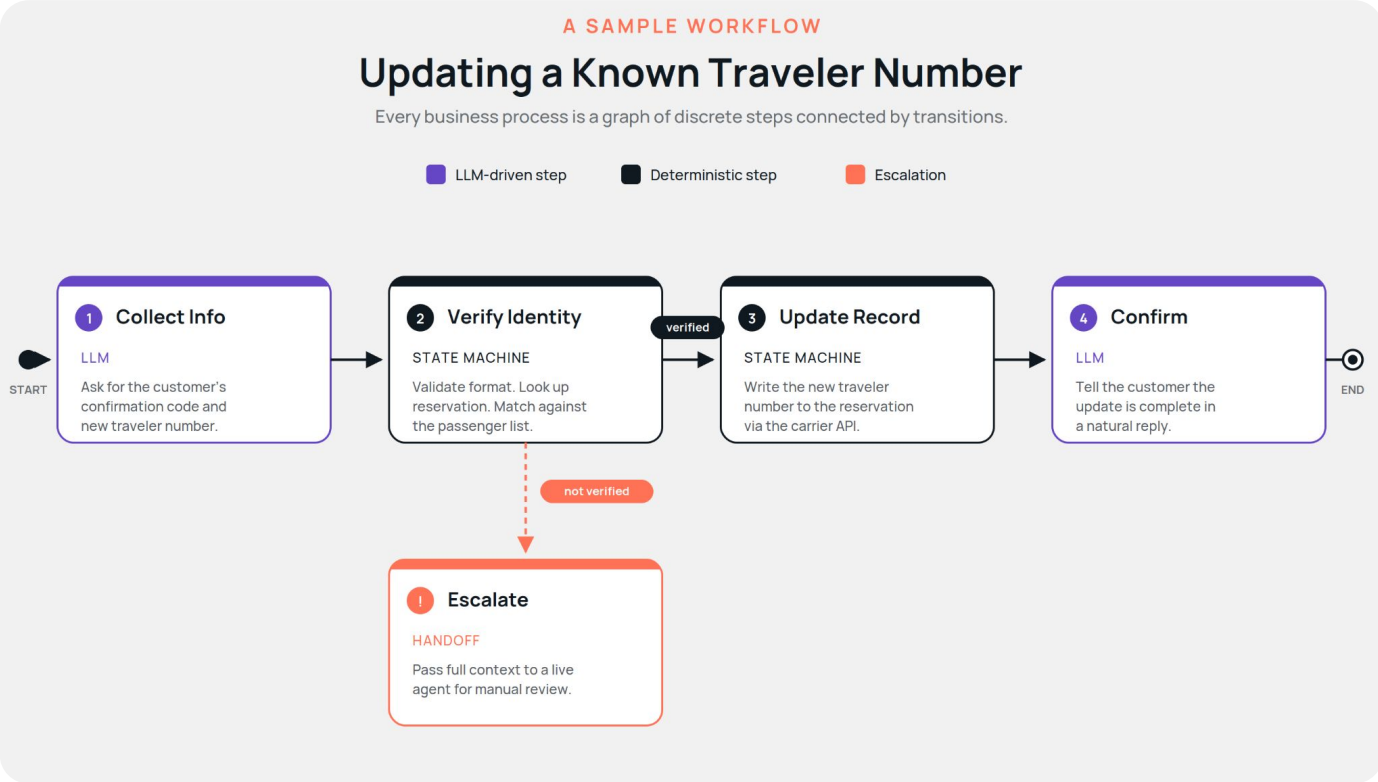
In Step-Based Flows, every business process is defined as a workflow: a graph of discrete steps (the individual states in the state machine) connected by transitions. Each step specifies:

- **What information the agent needs to collect** from the customer (e.g., first name, confirmation code, traveler number)
- **What instructions guide the agent's conversation** at that point (e.g., "Ask the customer for their confirmation code. It's a 6-character alphanumeric code found in their booking email.")
- **What transition logic executes** once the information is collected (e.g., validate the format, look up the reservation, match the customer's identity)
- **What conditions determine the next step** (e.g., if the reservation is found and the customer is verified, proceed to the update step; if not, escalate)

A collaboration between LLM and state machine

Here's what happens during the interaction:

1. **The customer says something.** "I need to update my Known Traveler Number on my reservation."
2. **The LLM recognizes the intent** and initiates the appropriate workflow. It sees a curated list of available workflows with descriptions—not an overwhelming set of tools.
3. **The workflow activates its first step.** The LLM receives focused instructions for just this step, not the entire process, along with a clear specification of what information to collect. The LLM asks the customer naturally for the needed details.
4. **The customer provides the information.** The LLM collects the data and signals the workflow to proceed.
5. **The state machine takes over.** Transition logic executes in a defined sequence: validate the traveler number format, call the reservation API, match the customer against the passenger list. The LLM is not involved in any of this. It doesn't decide which API to call, how to format the request, or how to interpret the response.
6. **The state machine evaluates transition conditions** and advances to the next step—or, if all remaining steps require no customer interaction, chains through them automatically without returning to the LLM at all.
7. **The LLM presents the result** to the customer in natural language, informed by what the state machine accomplished.



The critical point: the LLM participated in steps 1–4 and step 7: understanding the customer, collecting information, and communicating the result. Steps 5 and 6 were entirely deterministic. The LLM never constructed an API call, never decided the order of operations, and never evaluated whether conditions were met. It couldn't skip a validation because it never had the opportunity to.

Focused context, Not cognitive overload

At any given moment, the LLM sees only what's relevant to the current step: the instructions for that step, the specific fields it needs to collect, and knowledge base articles selected for that step. It doesn't see instructions for other steps, tools for other workflows, or the full complexity of the business process.

This is a fundamental difference from prompt-based approaches, where the model must hold the entire process in context at all times. In Step-Based Flows, the model's per-turn task stays simple and consistent regardless of how complex the overall workflow is. A system with three workflows and ten steps presents the same cognitive demand to the LLM as one with thirty workflows and two hundred steps.

Complexity scales in the state machine, not in the model's workload.

Channel-agnostic by default

Workflow definitions operate at a level of abstraction above the delivery channel. A single definition produces consistent behavior in both text-based chat and voice conversations. The core logic (step sequencing, data validation, API orchestration, transition conditions) applies regardless of whether the customer is typing or speaking.

When the interaction does need to differ between channels, the workflow supports channel-targeted branching within the same definition, so builders configure channel-specific paths where needed rather than maintaining entirely separate configurations. For example, the platform handles voice-specific interaction patterns like barge-in, interruption recovery, ASR handling, and DTMF support natively.

A workflow validated in one channel is inherently validated in the other for all shared logic, reducing the need for duplicate testing across channels.

The conversation feels human

Because the LLM's only job is to communicate, not to orchestrate, it can focus entirely on the quality of the interaction. The agent adapts its tone to the customer's mood, handles interruptions and tangents gracefully, and doesn't sound scripted because it isn't. The LLM generates every response naturally within the guardrails of its current step.

If a customer asks an unrelated question mid-workflow ("Actually, what's the baggage allowance for international flights?"), the agent answers from its knowledge base and picks up where it left off. Multiple workflows can even be active simultaneously, allowing the conversation to flow between topics without losing progress on any of them.

When a conversation does need a human, because the workflow reaches an escalation point, the customer asks for one, or the scenario falls outside coverage, the handoff follows the same deterministic logic. It's a structured workflow step, not an ad hoc bail-out. Context transfers intact, the reason for escalation is captured, and any required pre-handoff actions run automatically before the transfer.

Predictable execution

Every validation, API call, data transformation, and routing decision happens in code, not in the LLM's reasoning.

The state machine enforces step sequencing. The agent cannot issue a refund without first verifying account ownership, because the workflow structure makes it impossible. The agent never constructs API requests; transition logic handles the full integration pipeline, from mapping data to the right format through executing the call to parsing the response.

Because the LLM never reasons about business logic, interprets API responses, or decides what actions the system should take, the categories of content it can fabricate are narrower than in prompt-based architectures. It can't misrepresent what an API returned or what the system did, because it was never involved in those operations. Where a prompt-based agent juggles 20–50 tools and a monolithic instruction set, the LLM in Step-Based Flows sees scoped instructions for the current step and a single tool to trigger the transition.

As a second layer of defense, the agent's responses are validated against retrieved knowledge base articles by a real-time safety layer before they reach the customer, catching ungrounded claims that the structural separation alone might not prevent.

Every conversation turn maps to a specific step in a specific workflow version. When a supervisor reviews a transcript, they can see exactly where the agent was in the process, what instructions it was following, and what transition logic executed.

Prescriptive exception handling

No production system operates exclusively on the happy path. APIs time out. Customers provide data in unexpected formats. Edge cases surface that the workflow author didn't anticipate.

Because the execution path is defined in the workflow graph, failure handling is structural rather than ad hoc. Transition conditions can route to different steps based on error states: if a reservation lookup fails, the workflow can branch to a retry step, a manual-lookup step, or an escalation, depending on the nature of the failure. The LLM doesn't need to decide how to handle the error; the state machine routes it to the appropriate next step, and the LLM receives instructions appropriate for that context.

Verbatim delivery for regulatory compliance

Some interactions require absolute precision. Regulatory disclosures must be read verbatim. Compliance statements can't be paraphrased. Terms and conditions must be delivered word-for-word and acknowledged before proceeding.

Step-Based Flows handles this through a dedicated capability for verbatim delivery. When a workflow reaches a step that requires exact text, the state machine delivers the precise language directly, bypassing LLM generation entirely. The text is injected into the conversation history so the agent can naturally answer follow-up questions about it, but the original delivery is deterministic and exact.

The customer cannot proceed without completing the acknowledgment step. If they interrupt, the system can resume from where it left off or restart, depending on how the workflow is configured. This isn't the LLM "trying to remember" to finish reading a disclaimer; it's the state machine making it structurally impossible to skip.

Prompt-based approaches cannot provide deterministic guarantees for this kind of enforcement. The LLM may deliver the text correctly most of the time, but there is no structural mechanism ensuring it always will. In Step-Based Flows, completion and acknowledgment are enforced by the state machine, not by the model's compliance with instructions.

Built for builders

Step-Based Flows includes a dedicated workspace for the teams who build and maintain workflows, designed for CX designers and operations teams.

Visual workflow builder

Workflows are authored in a visual graph interface. Each step is a node, each transition is a labeled edge, and the entire process is visible at a glance. Selecting a step opens its configuration: the instructions that guide the agent's conversation, the information it needs to collect, the transition logic that executes, and the conditions that determine what happens next.

While any new system integration will need technical configuration, building and modifying workflows generally requires very little if any, code—the workflow structure, step instructions, data collection requirements, and transition conditions and knowledge base linking are all no-code, configured through the interface. The automated actions within a step—data validation, API integration, response parsing—are configured through ASAPP API Connections, supporting REST APIs, MCP, and custom code-based methods for non-standard protocols.

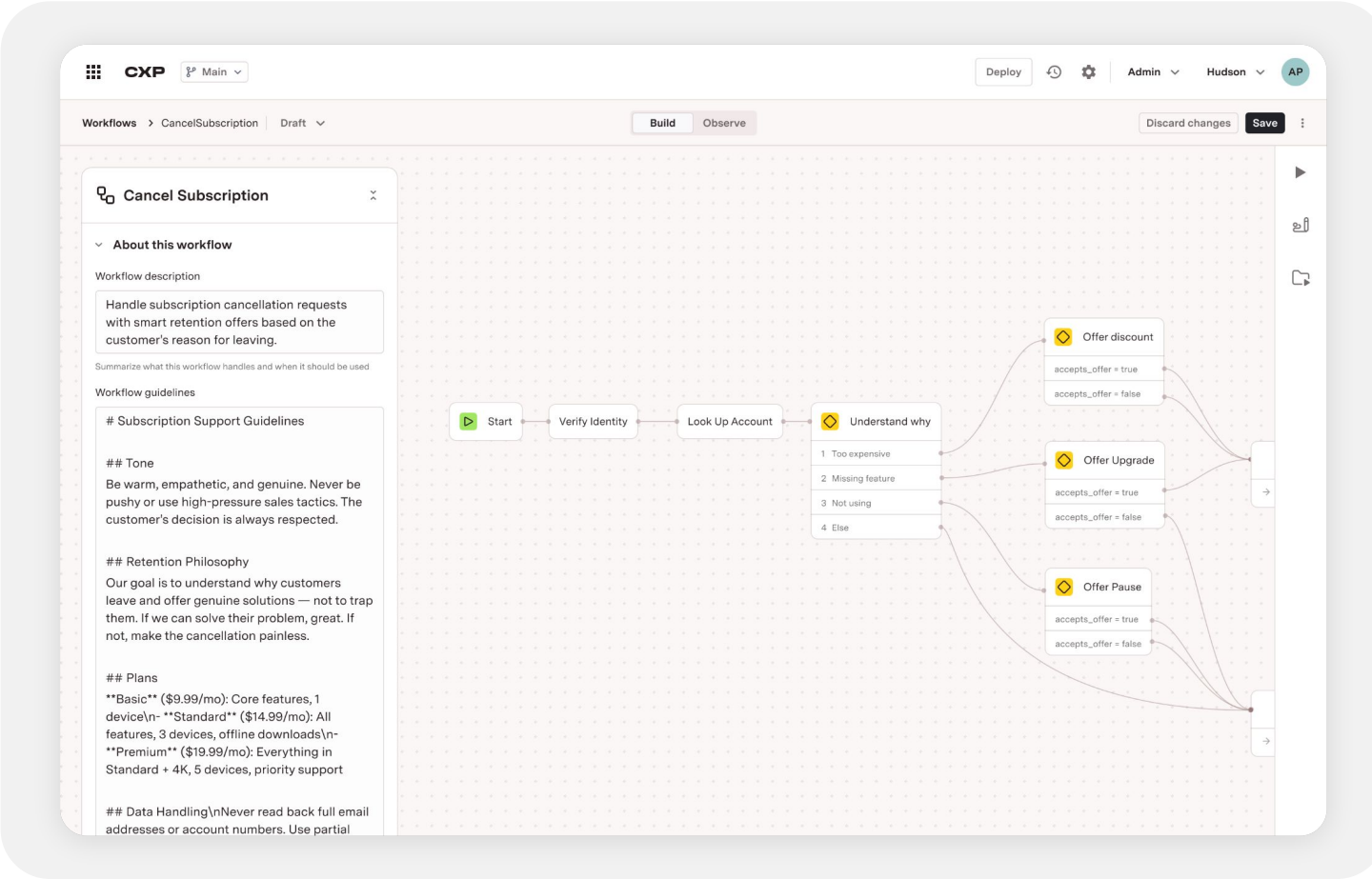


Figure 3. Visual workflow builder within ASAPP CXP.

Integrated testing

Testing is embedded directly in the authoring experience. Builders can:

- **Preview conversations** against draft workflows without deploying, watching in real time as the conversation progresses through the graph with the active step highlighted on the canvas
- **Create and run scenario tests** that validate specific paths through a workflow, with step-level diagnostics showing exactly which step a test failed at and why
- **Iterate rapidly:** make a change, run the affected tests, and confirm the fix without switching contexts. This is the same testing suite that makes model upgrades fast and low-risk: run your full scenario suite, identify any regressions, and address them before deployment.

Safe, auditable deployments

Workflows move through a governed deployment pipeline (draft → approval → production) with role-based access controls, immutable version snapshots, and the ability to diff and roll back changes. Every conversation turn in production traces back to a specific workflow version, creating a complete audit trail.

The workflow definition also governs what the LLM has access to within each step. When a node calls an external API, workflow builders configure which fields from the response reach the LLM's context and which are consumed entirely within transition logic—used to drive routing decisions without surfacing to the model.

Because these decisions live in the workflow definition itself, they are versioned and reviewable alongside everything else in the deployment pipeline. For enterprises in regulated industries, this makes the question of what the LLM was and wasn't given access to an auditable matter of record, not a behavioral inference from conversation logs.

Workflow configurations are validated before deployment: the system checks that every step can receive the data it needs, that types are consistent, and that transition conditions reference fields that actually exist. An entire class of configuration errors is caught before they reach production.

Step-level analytics

Because every conversation passes through named, discrete steps, production performance data can be anchored to specific points in the workflow graph. In observation mode, metrics are overlaid directly on the canvas: each step displays its containment rate, latency, and issue frequency as a color-coded heat map. Builders can see exactly where in a workflow customers are getting stuck, escalating, or experiencing errors—then click through to review the specific conversations driving those numbers.

This transforms troubleshooting from "something is wrong with the agent" to "step 4 of the reservation update workflow has a 15% escalation rate—let's look at those conversations and figure out why." Problems are localized to a specific point in a specific process—and fixes can be measured at the same granularity.

Most agent platforms offer conversation-level metrics at best. The ability to pinpoint a specific step in a specific workflow is a fundamentally different—and significantly better—operating model for CX teams.

Model flexibility and future-proofing

Step-Based Flows fundamentally changes what the LLM needs to be good at. In a prompt-based architecture, the model must excel at tool reasoning, multi-step planning, and complex instruction following—capabilities that vary significantly across models and model versions. This creates a tight coupling: your agent's reliability depends on specific model capabilities, and model changes can break behavior in unpredictable ways.

In Step-Based Flows, the model's job is simpler and consistent: follow the current step's instructions, collect specific information from the customer, and respond naturally. The complex reasoning (API orchestration, data validation, business logic, state management) happens in the state machine.

In practice:

- **Faster response times: no dependency on extended thinking.** Because the model isn't reasoning through multi-step tool chains, it doesn't need chain-of-thought or extended reasoning to perform reliably, removing a significant source of per-turn latency. The agent responds at conversational speed, not "thinking" speed.
- **No model vendor lock-in. Model selection is driven by conversational quality:** tone, empathy, fluency, rather than by the ceiling of the model's reasoning capability. This opens up a wider range of models for production deployment and avoids locking your agent's performance to a single vendor's model roadmap.
- **Resilience to pace of model evolution: business logic survives model changes.** Because workflows encode your processes in a deterministic state machine rather than in prompt engineering, model upgrades don't require redesigning your workflows. Integrated scenario testing makes the validation process during model transitions fast and thorough—workflow definitions can be expected to remain stable across model changes.

Conclusion

Remember the agent that forgot to verify account ownership before issuing a refund? That skipped a validation, hallucinated a policy detail, called the wrong API? Those failures happen because the LLM is responsible for both the conversation and the business logic—its reasoning stretched across tasks it shouldn't own. Step-Based Flows eliminates that class of problem structurally: the LLM thinks about the customer, a deterministic state machine thinks about the process, and neither is responsible for the other's job. The agent can't skip a verification because it never controls the sequence. API requests are constructed and executed by the state machine, not the model. And with the LLM's reasoning focused on understanding people rather than orchestrating processes, its cognitive energy goes where it matters—and the surface area for fabrication shrinks to a fraction of what prompt-based architectures expose.

For the teams building and maintaining these agents, the step-based structure changes the operating model. Problems are localized to specific steps in specific workflows. Fixes are targeted and testable. Changes don't break things they shouldn't.

Looking ahead, ASAPP is developing AI-assisted capabilities that accelerate how teams build on this foundation, including the ability to generate Step-Based Flow configurations from existing process documentation, knowledge bases, and historical conversation transcripts. Look for an upcoming white paper on AI-assisted workflow generation.

Learn more at asapp.com