The Shift to Agentic AI Workflows

This guide summarizes the key concepts, frameworks, and future trends for understanding and building agentic AI workflows, the next evolution beyond traditional automation.

By: Nate Herk

I. What Are Agentic Workflows?

Traditional automation (like n8n) requires you to drag nodes, configure each one, connect them, test, debug, and repeat. Every API connection, variable, and condition is on you.

Agentic workflows flip this model. Instead of telling the system *how* to do something, you tell it *what* you want in plain English. You explain where the data comes from, what needs to happen, and where it should end up. The agent figures out the rest.

The Analogy: Think of it like hiring a human developer. You wouldn't explain every line of code, you'd explain the outcome you want, the problem you're solving, the tools you're using, and what the end result should look like. Then they'd build it.

The Requirement: You don't need to know how to code, but you do need to communicate your plan clearly. If you can't explain what you want, neither a human nor an AI agent can build it. The good news is you can use the agent itself to brainstorm, it will ask all the questions it needs to build a robust solution.

II. The Four Key Changes in Agentic Workflows

1. Self-Healing

In traditional automation, when a workflow breaks, you read the error, tweak the node, test again, and repeat. This debugging loop consumes significant time.

With agentic workflows, the agent handles that loop for you. It tries something, runs it, checks what happens, and if something breaks, it figures out why, edits its own code and instructions, and updates everything so it doesn't make that mistake again.

You don't have to read or write any code. You just explain what "good" looks like and approve the changes it suggests.

2. Natural Language Control (For Real This Time)

Many tools claim to let you build with natural language (Lovable, Bolt, Lindy, n8n's AI builder), but they typically get you 60–70% of the way there and require significant manual cleanup.

The new generation is different. Instead of just taking your prompt and building, the agent first *interviews you* to ensure it knows everything it needs. It asks clarifying questions like:

- Who are the users?
- How many times should this run?
- What tools are you using?
- What happens if X occurs? What should happen if Y occurs?

Often, the agent asks questions you wouldn't have thought of until you ran into a problem months later.

Once built, natural language becomes the "remote control" for the whole automation. You can say things like "make that faster," "make it cheaper," "add a manual review step here," or "log all outputs to this Google Sheet", and it will just do it.

Parallel Agent Testing: You can have multiple agents running simultaneously. Ask one agent for five different approaches, then have five separate agents try each approach. Come back, stress test all solutions, and pick the cheapest, fastest, and highest-quality one.

3. Security

When agents write code you can't read, security becomes critical. The same large language models generating code can also constantly review it for security problems and vulnerabilities, on every single change.

Think of it like having a security-obsessed developer sitting next to you. Every time something changes, they double-check: Are API keys hidden? Is sensitive data being logged somewhere it shouldn't be?

You set the guardrails with natural language:

- "Never send customer phone numbers to any third-party tool."
- "Always stop this workflow if you exceed $5 of API usage."

Your job is to define what must never happen. The system's job is to enforce it.

4. Instant API and MCP Integrations

Traditional workflow building means wrestling with API authentication, headers, JSON bodies, query parameters, and reading pages of documentation just to get the request shape right.

With agentic workflows, you just say what tool you want to connect to: "Get my Fireflies transcripts, push them into ClickUp, and send me a Gmail summary." You provide your API keys, and the agent figures out the rest, it can search the web or go to the URL you provide to research the documentation on its own.

MCP (Model Context Protocol) is like an app store for agents, tools that are pre-wrapped and documented so agents know what's available and how to use each correctly. If a tool doesn't have an MCP yet, the agent can still read the API documentation and make the right request.

Because it's all code-based, it can handle retries, rate limits, pagination, and webhooks, things that can be painful to build manually.

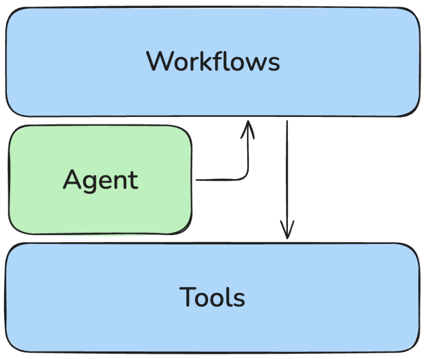
III. The WAT Framework

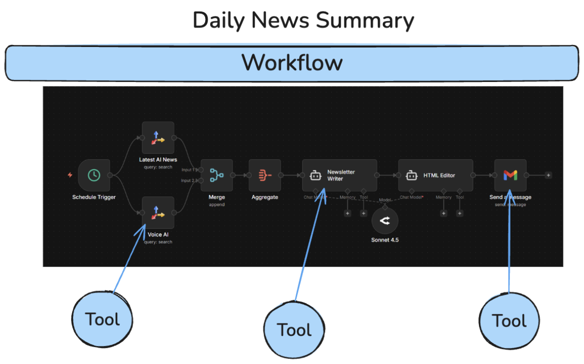
When building agentic workflows (demonstrated using Claude Code in VS Code), you work within a simple structure:

Component	Description
Workflows	Markdown SOPs defining *what* to do and *how*
Agent	The "brain" you talk to that helps figure out which workflows to execute
Tools	Python scripts for deterministic execution (scraping, outreach, exports, etc.)

Additional Files:

- **claude.md**, The system prompt/agent instructions for the whole project (what you're doing, how folders work, how to execute actions)





IV. Live Build Example: Lead Generation Automation

The Request: "Help me build a lead generation automation. I want to scrape dentists in Chicago so I can reach out and offer my AI automation services. Do research, scrape leads, write personalized outreach, and put all data in a Google Sheet."

The Process:

1. **Planning Mode:** The agent reads your files, familiarizes itself with the environment, and creates a plan.
2. **Question Phase:** The agent asks clarifying questions:
   - What data source should we use? (e.g., Google Places API)
   - How deep should enrichment go? (Basic vs. deep research)
   - What message tone? (Professional, friendly, premium)
   - Do you have required API keys set up?
3. **Execution:** Once questions are answered, the agent builds a to-do list and executes:
   - Creates tool files (scrape_dentist.py, generate_outreach.py, export_to_sheets.py)
   - Creates the workflow file (chicago_dentist_leads.md)
   - Wires everything together
4. **Output:** Each lead includes name, address, phone, website, rating, reviews, neighborhood, subject line, and a personalized outreach message referencing specific details (like "your 4.9-star rating from 714 reviews").
5. **Iteration:** Use natural language to refine: "Expand to California," "Make outreach more personalized," "Add email addresses," "Incorporate this PDF about my business."

Deployment: For automations that trigger on events or schedules without you present, you can publish scripts to platforms like Modal to run autonomously.

V. The Future of Agentic Systems

1. Fully Autonomous Workflows

Current workflows are reactive (waiting for webhooks, schedules, or form submissions). Next-generation agentic systems will be **proactive**, constantly scanning your CRM, inbox, and project management tools for inefficiencies, roadblocks, or risks.

They'll spot deals going cold before you notice, flag projects falling behind before deadlines are missed, and not just alert you, they may propose fixes or take action on their own.

Market Projections:

- 25% of enterprises using generative AI will deploy agentic pilots in 2025
- 50% by 2027
- By 2028, agents will act as autonomous partners handling complex multi-step problems
- AI agent market projected to grow from ~$8B (2025) to $40–50B by 2030 (43% CAGR)

---

2. Agents Managing Other Agents

We're moving toward teams of specialized agents: an email agent, research agent, reporting agent, data cleanup agent, and a **manager agent** that delegates work, reviews output, and stitches everything together.

Research shows multi-agent setups often outperform a single model by distributing tasks among specialists. Major companies are preparing for agent teams embedded across sales, support, operations, and finance, all coordinating with each other.

---

3. Protocols Like A2A (Agent to Agent)

MCP = How agents talk to tools
A2A = How agents talk to other agents

In April 2025, Google Cloud announced A2A as an open standard so AI agents from different vendors can communicate, share context, and coordinate across systems.

A2A defines:

- **Agent Cards**, Descriptions of what each agent can do
- **Shared Task Lifecycle**, How agents hand off work
- **Secure Context Sharing**, How agents share information safely

One agent can find candidates, another handles scheduling, another does background checks, all coordinating without you playing middleman.

Partners include: Salesforce, SAP, ServiceNow, Workday, and 50+ enterprise partners.

---

4. Long-Running Project Agents

Current agents are great sprinters but poor long-distance runners. They can crush one-off tasks, but managing projects for weeks leads to forgetting, hallucination, and drift.

Benchmark Example: VendingBench asks LLMs to run a vending machine business over time. Even strong reasoning models show high variance and get worse, forgetting orders, mistracking inventory, falling into loops.

Emerging Solutions:

- **Continuous Loops:** Plugins like Ralph Wiggum for Claude Code keep looping tasks until success conditions are met, with guard

---