AGENTIC WORKFLOWS 2026 GUIDE

From Beginner to Pro

Core Concepts

- PTRMO Cycle
- DOE Architecture
- Antigravity Workspaces
- Self-Annealing Agents

A comprehensive reference guide for building reliable, revenue-generating AI agents.

Generated by Gemini
Based on the Transcript of Nick Saraev's 6-Hour Course

1 Introduction

Agentic workflows represent the next evolution of automation. Unlike linear automations (e.g., Zapier or Make) that break when variables change, **Agentic Workflows** use Large Language Models (LLMs) to "reason" through problems, handle errors, and execute complex tasks dynamically.

This guide covers the core frameworks, tools, and structures required to build robust AI agents that function not just as chatbots, but as autonomous employees.

Linear vs. Agentic

Linear Automation: If A happens, do B. If B fails, the system crashes.
Agentic Workflow: Here is a Goal. Figure out how to get from A to Z. If B fails, try C.

2 The PTRMO Framework

To understand how an agent "thinks," we use the **PTRMO** cycle. This is the internal recursive loop an agent runs through to complete a task effectively.

P – PLANNING

Definition: The agent analyzes the request and breaks it down into steps *before* taking action.
In Practice: The agent reads the Directive, assesses available tools, and creates a manifest.
- *Example:* "First I will scrape the website, then find the email, then draft the message."

T – TOOLS

Definition: The specific capabilities the agent has access to. Unlike a chatbot, an agent has "hands."
In Practice: These are Python scripts or API connectors.
- *Examples:* A calculator, a web scraper, an email sender, a database connector.

R – REFLECTION

Definition: The ability to critique its own work.
In Practice: After generating an output, the agent pauses to ask: "Does this meet the criteria in the Directive?" If not, it self-corrects before the user ever sees the result. This is vital for **Self-Annealing**.

M – MEMORY

Definition: Maintaining context over time.
- **Short-term:** The current conversation context window.
- **Long-term:** External files (e.g., `clients.json`, `knowledge_base.md`) where data persists.

O – ORCHESTRATION

Definition: The "Brain" that manages the process.
In Practice: The Orchestrator decides *which* tool to use and *when*. It routes data between the Plan, Tools, and Memory.

3 The DOE Architecture

The **DOE Framework** is the architectural standard for building reliable agents. It separates the workflow into three distinct layers to prevent hallucinations and ensure consistency.

3.1 D - Directive (The "What")

- **Role:** The Manager / The Boss.
- **Format:** Typically a Markdown file (e.g., `directive.md` or `agents.md`).
- **Content:** Context, Goals, Constraints, and SOPs (Standard Operating Procedures).
- **Golden Rule:** NO CODE lives here. Only natural language instructions.

3.2 O - Orchestration (The "Who")

- **Role:** The Employee / Decision Maker.
- **Format:** The LLM Loop (e.g., Claude 3.5 Sonnet, GPT-4o).
- **Function:** Reads the Directive, calls Execution tools, evaluates output, and loops.

3.3 E - Execution (The "How")

- **Role:** The Tools / The Hands.
- **Format:** Deterministic Code (Python scripts).
- **Why Code?** LLMs are probabilistic (they guess). Code is deterministic (it is exact).
- **Example:** Don't ask an LLM to reverse a list of 500 names (it might fail). Have the LLM call `reverse_list.py`.

DOE Workflow Summary

Directive (Manager) → gives orders to → **Orchestrator** (LLM)

↓

Orchestrator calls → **Execution** (Python Script)

↓

Result returns to → **Orchestrator** for Reflection

4 Workspace & Antigravity

4.1 The "Antigravity" Concept

"Antigravity" refers to a specific IDE setup designed for rapid iteration. It allows you to edit the Directive and run the Orchestrator instantly while seeing the agent's "thought process" (logs) in real-time.

4.2 Essential VS Code Extensions

To replicate the Antigravity workflow, install these extensions:

1. **Python (Microsoft):** Essential for running Execution scripts.
2. **Pylance:** For code intelligence and debugging.
3. **Markdown All in One:** For editing `Agents.md` and Directives efficiently.
4. **Prettier:** To keep JSON data and code clean.

4.3 Workspace Structure

A professional Agentic Workspace should be organized as follows:

```
/Project_Name
|
|-- /directives          <-- (D) The Managers
|   |-- outreach_sop.md
|   |-- research_guidelines.md
|
|-- /orchestrator        <-- (O) The Brain
|   |-- main.py          (The loop calling the LLM)
|   |-- agent_config.py
|
|-- /execution           <-- (E) The Tools
|   |-- tools.py         (Function definitions)
|   |-- /scripts
|       |-- scraper.py
|       |-- email_sender.py
|
|-- /memory              <-- (M) Storage
|   |-- leads_db.json
|
|-- .env                 (API Keys)
```

5 Files & Skills

5.1 The Agents.md File

This is the single most important file in your workspace. It serves as the dynamic "System Prompt."

- **Purpose:** Defines the agent's persona and rules.
- **Key Section - Capabilities:** You must explicitly list what the agent *can* and *cannot* do.
- **Example:** "You have access to a tool called `get_stock_price`. Use this whenever the user asks for financial data."

5.2 Defining Skills (The Execution Layer)

Skills are discrete blocks of code the agent can wield. Most LLMs require skills to be defined in a JSON Schema so they know exactly what arguments to pass.

Listing 1: Example Skill Definition (Python)

```python
# In tools.py
def get_website_content(url):
    """
    Scrapes the text content of a given URL.
    Args:
        url (str): The website to scrape.
    """
    # ... python code using BeautifulSoup ...
    return text
```

6 Real-World Examples

6.1 Example A: The Lead Enrichment Agent

- **Directive:** "Take this list of domains. Find the CEO's name and their recent LinkedIn post. Generate a personalized icebreaker."
- **Orchestration:** Agent reads domain 1 $\rightarrow$ Calls `scrape_linkedin` tool.
- **Execution:** `scrape_linkedin.py` runs (using a proxy) and returns raw text.
- **Reflection:** Agent checks if the text contains a recent post. If yes, it writes the icebreaker.
- **Output:** Saves to `leads_enriched.csv`.

6.2 Example B: The Proposal Generator

- **Directive:** "Read the transcript of the sales call in `memory/call_log.txt`. Write a proposal addressing specific pain points."
- **Orchestration:** Agent identifies pain points (e.g., "manual data entry").
- **Execution:** Agent uses a `generate_pdf` tool to fill a template.

7 Conclusion: Self-Annealing

The “Pro” level of agentic workflows is **Self-Annealing** (Self-Healing).

The Problem: APIs fail, websites change structure, and code breaks.

The Solution: Build a `try/except` loop into the Orchestration layer.

How it works:

1. Agent tries to scrape a site.
2. Tool returns `Error: 403 Forbidden`.
3. Instead of crashing, the Agent reads the error.
4. Agent “reasons”: “I was blocked. I should try again using a different proxy.”
5. Agent calls the tool again with new parameters.

By combining **PTRMO** logic with the **DOE** structural framework, you create agents that are not just chatty, but capable, reliable employees.