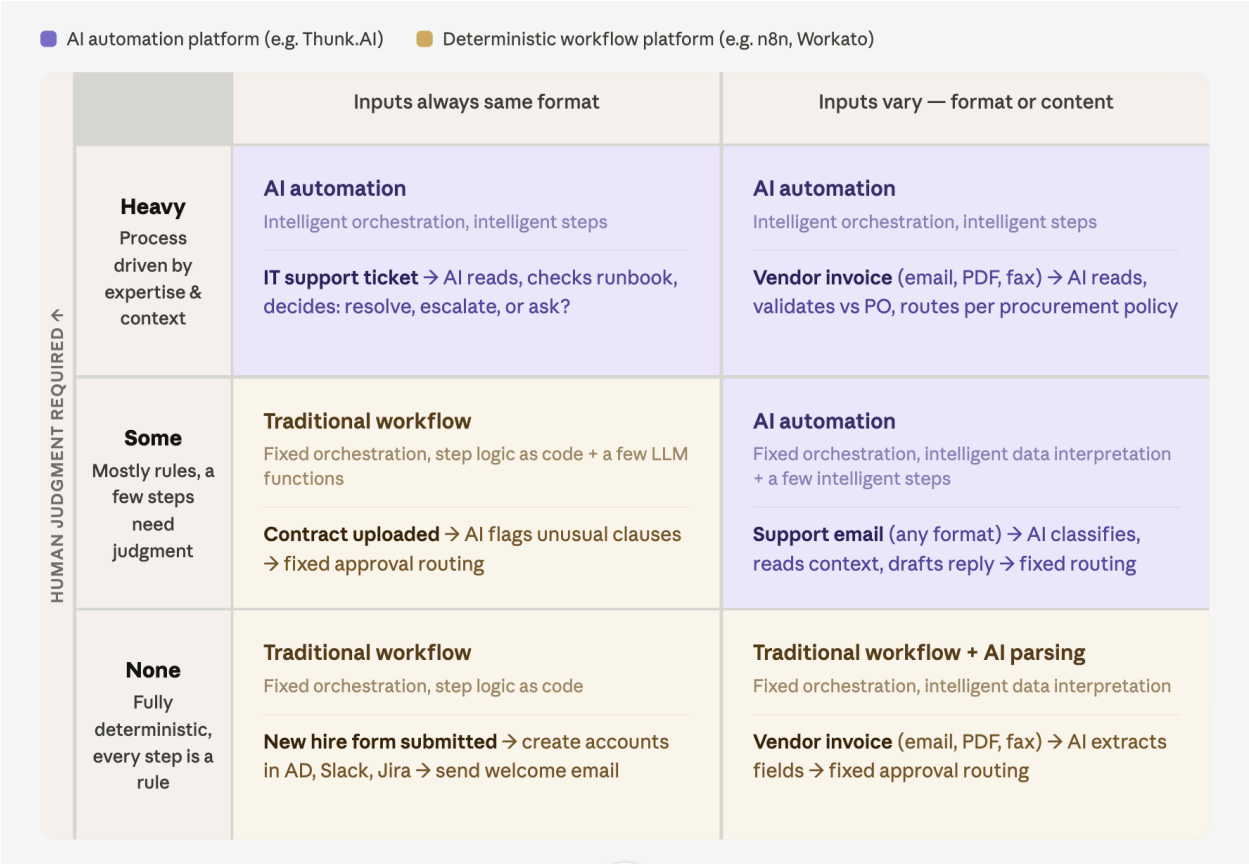
# AI Agentic Automation: Code vs No-Code

*Trade-offs between different approaches to building Agentic automation*

There are many productivity benefits to automating business processes. However, it is not easy to build these automations. The legacy approach to business process automation was to use deterministic workflow automation products like Workato, Make.com, n8n, etc. These work well when the inputs are consistent and the instructions are complete and deterministic. In practice, however, there are many unexpected inputs, errors, and edge cases that have to be handled. Additionally, there is a whole class of business processes that are more subjective and flexible, where there is a standard operating procedure but it is not translatable into a detailed "code-level" workflow. Hence the current interest in intelligent **"AI Agentic"** automation.

## Deterministic Workflows vs Agentic Workflows

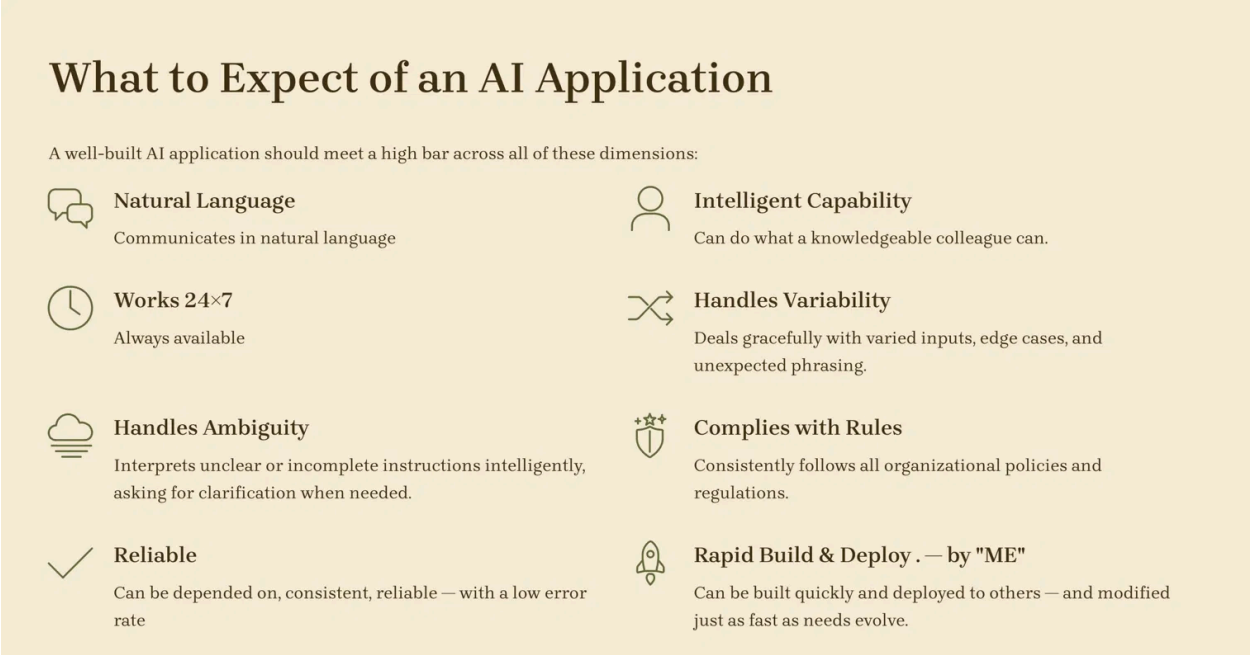
The two models (deterministic workflows and Agentic workflows) are endpoints on a continuum. The appropriate choice is a function of two independent variables: the variability of the *inputs* (from standardized and structured to free-form and heterogeneous) and the nature of the *process logic* (from a fixed deterministic procedure to logic specified in natural language and intended to be applied across a range of unanticipated situations). Plotting those axes yields a small number of regimes:




- **Deterministic logic, structured inputs.** A conventional program or traditional workflow platform (e.g. n8n) is appropriate.
- **Natural-language logic and/or variable inputs.** When the process must be applied sensibly across realistic variation, both orchestration and individual steps need to be intelligent. This requires Agentic automation.
- **Partially deterministic logic and data**: either approach might be the most appropriate. Traditional workflow platforms are adding intelligent data-interpretation steps, and AI automation platforms are adding mechanisms to make orchestration more deterministic and controllable.

## Requirements for Agentic Workflows

Agentic workflows are an important category of AI applications. Like all AI applications, they have a set of common requirements: it should communicate in natural language, handle variability and ambiguity gracefully, comply with organizational rules, and be reliable enough to depend on — while remaining fast and natural to build and modify as needs evolve.




## How to build Agentic Workflows

How should Agentic automation be built? There are two basic approaches described in this article:

1. Using traditional code (either programming directly against LLM APIs like OpenAI's GPT API, or using a developer platform for Agentic applications like LangGraph)
2. Using a no-code platform (like Thunk.AI).

This article compares these implementation strategies on the dimensions that determine total cost and time-to-production. The running example is accounts-payable invoice processing, but the general analysis applies to any enterprise Agentic workflow.

Many business processes lend themselves to the same structural pattern: read a ticket from one system, interpret an unstructured or semi-structured document, validate the extracted data against a set of constraints, take some appropriate actions, and then write the result into another system. These processes resist conventional automation because their inputs vary in format and content and it is hard to deterministically define the control logic for all cases ahead of time.

In a representative accounts-payable flow, invoices are managed in ServiceNow, which captures each invoice, logs it, and opens a payment case. Creating the ticket triggers a workflow that passes the invoice through Intelligent Document Recognition (IDR), decides on how to process the invoice, and forwards the extracted data and decision to Oracle Financials for processing and payment. Topologically it is a four-stage pipeline: **ServiceNow ticket → IDR/OCR extraction → Decision -> Oracle Financials**.

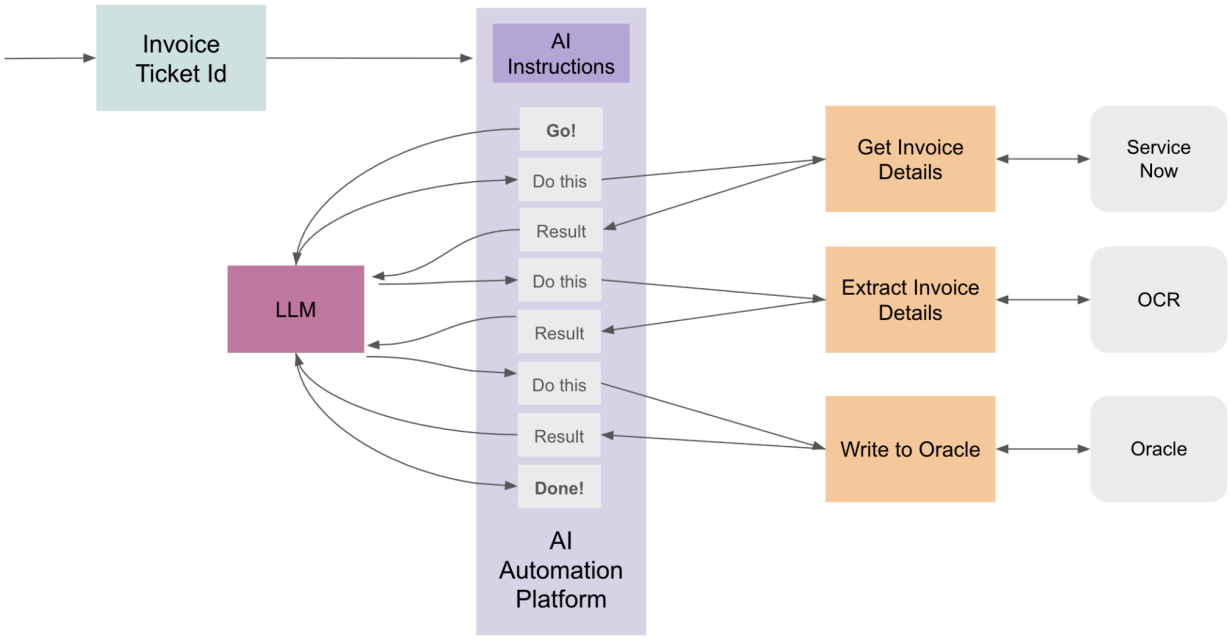
The target behavior is well-defined at a high level. However, the work at each stage and each stage boundary is a potential failure point. The observed failure modes fall into five categories:

- **Extraction errors** — OCR/IDR misreads fields or returns low-confidence values for non-standard invoice layouts.
- **Validation errors** — extracted values fail to reconcile against required fields, reference data, or business rules.
- **Decision errors** — incorrect decisions made on actions to take.
- **Transmission errors** — the write to Oracle does not occur, partially completes, or posts incorrect data.
- **Manual fallback** — each of the above is caught (if at all) by a human, adding latency and rework and breaking end-to-end traceability.

## The Architecture of Agentic Workflows

An Agentic workflow has the same external skeleton but replaces hard-coded control flow with a model that makes decisions and takes actions intelligently at runtime. The process intent is specified in natural language, and the AI agent operates as a controller at every step of the high-level workflow. For each such high-level step, it repeatedly chooses a tool to call, reads the tool result, updates its working state, and selects the next action — repeating until a termination condition is reached.

Concretely, a high-level orchestration engine moves the workflow through its four steps. At each step, the Agentic harness loop is *initialize* → *select action* → *invoke tool* → *observe result* → *select next action* → *terminate*. The Agentic harness executes tool calls that can include API requests to ServiceNow, the OCR engine, and Oracle.



Because action selection is dynamic and happens at runtime rather than being fixed at build time, the workflow can interpret variable inputs and recover from conditions that a deterministic flow would not have enumerated. The trade-off is that this very flexibility leads to a lack of determinism and makes reliability harder to guarantee.

## Failure Modes for Agentic Workflows

Moving an Agentic workflow from a demonstration to a system that meets a production-level reliability bar is a difficult engineering problem. It may fail for five distinct reasons, each requiring a different mitigation. A platform intended for production must address all five:

1. **Instruction errors.** The natural-language specification is ambiguous, internally inconsistent, or omits cases that the agent then resolves incorrectly.
2. **Configuration errors.** The workflow lacks the data, credentials, schema, or tool definitions required to act, or has them wired up incorrectly.
3. **LLM errors.** The AI model reasons incorrectly, hallucinates values, or selects an inappropriate action given correct inputs.
4. **Orchestration errors.** The execution layer mismanages state, context, retries, concurrency, or termination — independent of the model's output.
5. **Tool errors.** An integration (ServiceNow, the OCR engine, Oracle) is unavailable, rate-limited, or returns malformed or incorrect data.

These error modes explain a common observation: a demo of an Agentic workflow is weak evidence of production readiness. A demo exercises the happy path, where instruction, configuration, model, orchestration, and tools all behave. Production deployments reveal the long tail of situations where they do not. Reported figures put the share of Agentic automation projects that reach a production-grade reliability bar in real environments at roughly **5–10%**.

Only one of these five failure categories — LLM errors — is a property of the AI model and improvements in foundation models helps mitigate these errors. The remaining four failure categories are properties of how the workflow is specified, configured, orchestrated, and integrated. They are characteristics of the *execution platform and its surrounding tooling*, not of the underlying AI model.

## Implementation Strategies for Agentic Workflows

There are two common strategies for implementing an Agentic workflow:

1. Implemented by software engineers using code
    a. Fully custom development using LLM APIs
    b. Using a developer framework such as LangGraph
2. Implemented by process owners using a no-code platform,

While the code-based approach has no constraints on expressive power, the no-code approach is limited to the expressive power of the particular platform chosen. They also differ on many

other dimensions: **which party owns the reliability concerns above**, how intent is specified, maintenance, and the resulting time-to-production and cost-scaling characteristics.

| | Custom development | Developer framework (e.g. LangGraph) | No-code platform (e.g. Thunk.AI) |
|---|---|---|---|
| **Do you need software engineers?** | Yes | Yes | **No, but some platform training** |
| **How is intent specified?** | Ad-hoc in code | Ad-hoc in code | **Application model + authoring environment** |
| **Who detects / compensates for errors?** | Your developers | Mostly your developers | **The platform** |
| **Who owns security & compliance?** | Your developers | Mostly your developers | **The platform** |
| **Who owns deployment & live execution?** | Your developers / IT | Your developers / IT | **The platform** |
| **Who incorporates AI tech updates?** | Your developers | Mostly your developers | **The platform** |
| **Time to initial demo** | A few weeks → a week | A week → a day | **A day → an hour** |
| **Time to reliable production** | 1 year – never | 1 year+ (<10% make it) | **~3 months** |
| **Scaling cost across many workflows** | Linear | Linear | **Sub-linear** |
| **Expressive power** | Rich | Rich | **Depends on the platform** |

Custom development and developer frameworks both require software engineers and assign the error-handling, security/compliance, deployment, and AI-currency concerns to the implementing team. Their cost scales *linearly* in the number of workflows, since most of that work is repeated per workflow. A no-code platform approach moves those concerns into the platform: intent is expressed through an application model and authoring environment rather than ad-hoc code, error detection and compensation are handled by the runtime, and the marginal cost of an additional workflow is lower, giving *sub-linear* scaling across a portfolio of processes.

The time-to-production figures follow from where the reliability work lands. Custom code builds typically reach a demo in one to two weeks but a reliable production system takes about a year, if at all. Framework-based code builds reach a demo faster but still commonly take a year or

more, with under 10% reaching production. A no-code platform approach reaches a demo in hours and a production deployment on the order of three months, because the platform — not the project team — supplies the error-handling, deployment, and maintenance machinery. The platform however does constrain the Agentic workflows to conform to the expressive power supported.

## Summary

Agentic automation extends the reach of business automation to processes with variable inputs and natural-language logic. ROI, however, is realized only when automations achieve production-grade reliability. The dominant variable in whether an Agentic workflow reaches production is the implementation/execution platform and surrounding tooling, not the choice of foundation model.

No-code AI automation platforms like Thunk.AI handle the critical failure modes allowing each workflow implementation to focus on the process intent. As a consequence, they can be implemented and modified rapidly and by process owners rather than software engineers. **When time and cost are important factors and if the no-code platform can express the process logic, the no-code platform is the sensible choice**.

Code-based implementation approaches are more expensive as they have to handle the full complexity of Agentic harnesses for each workflow. On the other hand, they enable a full range of implementation choices without constraints. **When customizing the Agentic harness for specialized outcomes is critical, and time/cost are less important factors, the code-based approach is the sensible choice**.

## When to use Thunk.AI

HUMAN JUDGMENT REQUIRED ↑

| | Inputs always same format | Variable data — different formats, flexible content |
|---|---|---|
| **Heavy**<br>Process driven by expertise & context | **YES**<br>Intelligent orchestration, intelligent steps<br>**IT support ticket** → AI reads, checks runbook, decides: resolve, escalate, or ask? | **YES**<br>Intelligent orchestration, intelligent steps<br>**Vendor invoice** (email, PDF, fax) → AI reads, validates vs PO, routes per procurement policy |
| **Some**<br>Fixed process, one or two steps need judgment | **NO**<br>Fixed orchestration, step logic as code + a few LLM functions<br>**Contract uploaded** → AI flags unusual clauses → fixed approval routing | **YES**<br>Fixed orchestration, intelligent data interpretation + a few intelligent steps<br>**Support email** (any format) → AI classifies, reads context, drafts reply → fixed routing |
| **None**<br>Fully deterministic, every step is a rule | **NO**<br>Fixed orchestration, step logic as code<br>**New hire form submitted** → create accounts in AD, Slack, Jira → send welcome email | **MAYBE**<br>Fixed orchestration, intelligent data interpretation<br>**Vendor invoice** (any format) → AI extracts fields → fixed approval routing<br>*Mix of deterministic and AI logic as appropriate* |