Agent Engineering

with Gemini and Antigravity

Book 1: Foundations

BUILD | REASON | ACT | VERIFY | DEPLOY

Venkatesh Tadinada

Head AI Consulting, CloudKarya, Inc.

Agent Engineering

with Gemini, ADK & Antigravity

Book 1: Foundations

Build an Inspectable AI Agent with the WidgetWare SDR Lab

Venkatesh Tadinada

Head AI Consulting, CloudKarya, Inc.

SAMPLE EDITION

CloudKarya Press • July 2026

Copyright

Copyright © 2026 Venkatesh Tadinada. All rights reserved.

Published by CloudKarya, Inc. under the CloudKarya Press imprint.

This Sample Edition is provided for personal reading and evaluation. It may not be resold, redistributed, uploaded to other websites, or incorporated into another publication without prior written permission from the publisher.

The complete Early Access Edition contains all eleven chapters of Book 1 and may be updated as explanations, examples, exercises, and code references improve. Purchasers of the full edition receive future updates through the distribution platform.

Product names and trademarks belong to their respective owners. Their use does not imply endorsement.

First Sample Edition, July 2026.

Typeset from the author's Markdown manuscript.

To Chitra—my better half, in more ways than one.

About This Sample Edition

This free Sample Edition introduces *Agent Engineering with Gemini, ADK & Antigravity* and the WidgetWare SDR Lab.

It contains the complete Preface, Introduction, and Chapter 1. Together, these sections explain why Agent Engineering is a distinct professional discipline, introduce the Seven Steps to Agent Engineering, define the WidgetWare SDR use case, distinguish models, assistants, workflows, agents, and agentic systems, and establish measurable boundaries for the system built throughout Book 1.

The complete Early Access Edition contains all eleven chapters, including the Antigravity engineering harness, Gemini context engineering, Google ADK agents, reusable Skills, structured contracts, tool engineering, MCP research, multi-agent workflows, human approval, evaluation, deployment, and bounded loop engineering.

Readers who register for the Community Edition receive an expanded free edition that also includes Chapter 2: *Building with Antigravity*.

Contents

Preface

Software engineering has always been shaped by changes in abstraction.

We moved from machine instructions to programming languages, from individual programs to operating systems, from physical servers to cloud platforms, and from manually configured infrastructure to declarative automation. Each transition changed not only what we could build, but also what engineers needed to understand.

Agent engineering is another such transition.

A language model can generate text, analyze information, write code, and reason about unfamiliar situations. Yet a useful enterprise system requires much more than a capable model. It needs the right context, controlled access to tools, reusable procedures, deterministic workflows, secure execution, observable behavior, and evidence that it performs reliably. Without those layers, an impressive demonstration remains only a demonstration.

This book is about those layers.

I have spent more than three decades building, architecting, teaching, and advising on enterprise technology. Across data warehousing, business intelligence, cloud computing, machine learning, and generative AI, one lesson has remained constant: the technology that attracts attention is rarely the whole system. The real work lies in turning promising capability into something dependable, inspectable, and useful.

That is the central argument of this book:

> *An agent is not merely a model with a prompt. It is an engineered system in which probabilistic intelligence operates inside deterministic boundaries.*

The distinction matters. A model may suggest what to do. Software must still decide what it is allowed to do, what data it may access, which tools it can call, how outputs are validated, when a person must approve an action, and how success or failure will be measured.

The title names three important parts of the development experience:

- **Gemini** supplies the model intelligence.
- **Agent Development Kit**, or ADK, supplies a code-first framework for agents, tools, sessions, workflows, and evaluation.
- **Antigravity** supplies the agent-assisted engineering environment in which we specify, inspect, build, test, and improve the system.

But the book is not simply a product tutorial. Products change. Interfaces evolve. Commands are replaced. The durable subject is **Agent Engineering**: the discipline of designing systems that combine model reasoning with context, tools, skills, workflows, evaluation, and guardrails.

To keep the discussion concrete, we will build one system throughout the book: the **WidgetWare SDR Lab**. WidgetWare is a fictional software company. Its Sales Development Representative process gives us a realistic mixture of open-ended reasoning and controlled business execution. The system must understand an ideal customer profile, research target companies, distinguish evidence from inference, qualify opportunities, draft relevant outreach, and stop for human approval before any external action.

This is intentionally not a fully autonomous sales machine. Autonomy is not the objective. Business value under controlled risk is the objective.

The lab also gives us a cumulative learning path. We begin with a bounded problem and a clean development workspace. We add context, then an ADK agent, reusable Skills, structured outputs, tools, evidence-backed research, multi-agent workflows, approval gates, evaluation, and deployment. At every stage, the reader should be able to inspect what changed and explain why it was added.

The intended reader does not need to be an advanced Python programmer. The implementation favors clear, conventional code over clever code. The deeper challenge is architectural judgment:

- When should reasoning be delegated to a model?
- When should behavior remain deterministic?
- What context is necessary, and what context is dangerous noise?
- Is a capability better represented as an instruction, a Skill, a tool, or another agent?
- How should agents exchange information without losing meaning?
- Where must human approval remain mandatory?
- What evidence would convince us that the system is improving?

These questions are more durable than any particular API call.

Book 1 focuses on foundations. By the end, you will have an inspectable, evaluated, deployable agent application. Book 2 will extend that application into a stateful and governed enterprise agent platform with persistent memory, enterprise retrieval, controlled loops, distributed collaboration, identity, security, observability, cost management, and continuous evaluation.

My hope is that this book helps readers move beyond two common extremes: fear that agents are uncontrollable, and excitement that ignores en-

gineering discipline. Both are incomplete. Agent systems can be useful and trustworthy when we design them with humility, explicit boundaries, and measurable expectations.

The model supplies intelligence. Engineering makes that intelligence usable.

— **Venkatesh Tadinada**
Head AI Consulting, CloudKarya, Inc.

Introduction: Building Intelligence Inside Boundaries

Why this book exists

The easiest way to build an agent demonstration is to connect a model to a prompt and a few tools. The easiest way to misunderstand agent engineering is to assume that the demonstration is the system.

A production-minded agent application must answer questions that a prompt alone cannot answer:

- What business objective is the agent responsible for?
- Which decisions may the model make?
- Which actions must remain deterministic?
- What information should be placed in context?
- Which tools may be used, and with what permissions?
- How are outputs validated before downstream systems consume them?
- How do multiple agents divide work and exchange results?
- When must a human approve the next step?
- How do we test a system whose outputs are not always identical?
- How do we observe, deploy, and improve it?

This book develops a practical answer through a cumulative system.

The reference system: WidgetWare SDR

WidgetWare is a fictional business-to-business software company. It needs help identifying and researching prospective customers. A human Sales Development Representative typically performs several related jobs:

1. Understand the company's products and ideal customer profile.
2. Identify a target account.
3. Research recent, relevant evidence about that account.
4. Determine whether the account appears qualified.
5. Explain the qualification decision.
6. Draft a personalized outreach message.
7. Obtain approval before contacting the prospect.

These activities are well suited to Agent Engineering because they contain both predictable steps and judgment-intensive work. Research and synthesis benefit from model reasoning. Validation, permissions, state transitions, and approval gates benefit from deterministic software.

The goal is not to replace every human activity. The goal is to construct a system in which the model performs the work it is good at while software and people retain control over high-risk decisions.

The Seven Steps to Agent Engineering

The book uses one seven-step framework, applied consistently across every technology this series covers. Gemini, ADK, and Antigravity are this edition's implementation; the steps themselves are vendor-independent, because they describe engineering activities, not product features.

1. Frame the Use Case

Define the business objective, the intended users, what the system may and may not do, and the autonomy level it is permitted before any code exists. This is a discipline, not a formality — Chapter 1's WidgetWare charter and acceptance criteria are this step made concrete.

2. Build Context

Assemble the information Gemini reasons over: system instructions, business rules, ideal-customer-profile data, retrieved evidence, and task-specific state, deliberately separated so each can be inspected, tested, and changed independently.

3. Design Agent Capabilities

Choose or build the Skills, structured contracts, and tools the agent needs to do real work — the reusable procedures, typed interfaces, and controlled integrations that turn "a model that can reason about this" into "a capability the system can actually invoke."

4. Build the Harness

Establish the operating environment the agent runs inside: Antigravity as the specification-driven development harness, and ADK's own runtime — agents, sessions, events, and permissions — as the execution harness the built system runs on.

5. Orchestrate Workflows

Sequence steps, coordinate specialized agents, manage state transitions explicitly, and gate progress on human approval before any external or consequential action.

6. Engineer Loops

Take a workflow that has already been evaluated on one case and wrap it in a bounded process that discovers the next unit of work, persists state across interruptions, and decides — explicitly — whether to continue, retry, stop, defer, or escalate. This is a discipline in its own right, distinct from orchestrating a single pass through a workflow, and it is where Book 1 closes.

7. Evaluate & Govern

Measure whether the system is actually good — not merely whether it runs — and keep it inside its boundaries once it is. Book 1 builds the evaluation half of this step in depth; production monitoring and governance, the other half, are Book 2's concern.

How Book 1 progresses

Book 1 is organized as a build sequence, and that sequence does not track the seven steps in numeric order — Chapter 2 builds the harness (Step 4) before Chapter 3 builds context (Step 2), because a reader needs a working Antigravity workspace before there is anywhere to put business rules. The steps describe a dependency structure among engineering concerns, not a fixed reading order.

- Chapter 1 establishes the discipline and defines the WidgetWare problem (Step 1).
- Chapter 2 creates the Antigravity development harness (Step 4).
- Chapter 3 designs model and context behavior (Step 2).
- Chapter 4 implements the first ADK agent (Step 4, continued).
- Chapter 5 extracts reusable Skills (Step 3).
- Chapter 6 introduces structured outputs and contracts (Step 3, continued).
- Chapter 7 adds controlled tools (Step 3, continued).
- Chapter 8 connects evidence-backed research through tools and MCP (Step 3, continued).
- Chapter 9 introduces multi-agent workflows and human approval (Step 5).

- Chapter 10 evaluates and deploys the integrated, single-account system (Step 7).
- Chapter 11 wraps that evaluated system in a bounded ADK loop that processes many accounts (Step 6) — Book 1's closing chapter.

Every chapter leaves the repository in a usable state. A reader should be able to begin from the previous checkpoint, complete the chapter, run the tests, and inspect the new capability.

What Book 1 deliberately does not do

Book 1 introduces production thinking without attempting to solve every enterprise concern. It does not deeply implement:

- long-term memory;
- production-scale retrieval-augmented generation;
- distributed Agent-to-Agent collaboration;
- enterprise Agent Registry and Gateway governance;
- network containment;
- comprehensive Model Armor policies;
- model-routing economics;
- production-scale observability; or
- continuous online evaluation.

These topics belong in Book 2 because they are easier to understand after the reader has built and evaluated one complete agent application.

A principle for the entire series

The book repeatedly asks a simple question:

> *Is this behavior better expressed as model reasoning or deterministic software?*

Use deterministic software when the required behavior is known. Use model reasoning when interpretation, synthesis, or adaptive judgment is required. Connect the two through typed contracts, explicit state, narrow tools, and measurable evaluation.

That balance is the foundation of inspectable agents.

Chapter 1: From Language Models to Agent Engineering

Chapter purpose

This chapter establishes the conceptual foundation for the book. The reader learns why a language model is not an agent, why an agent is not automatically a useful system, and how Agent Engineering combines probabilistic intelligence with deterministic control.

Learning objectives

By the end of this chapter, the reader should be able to:

- distinguish a model, assistant, workflow, agent, and agentic system;
- identify tasks that require reasoning versus deterministic execution;
- explain the autonomy spectrum;
- describe the seven engineering layers used in this book;
- define the WidgetWare SDR problem and its boundaries; and
- write measurable success criteria for the Book 1 system.

Seven-Step mapping

Primary: Frame the Use Case
Supporting: Evaluate & Govern

The WidgetWare increment

Create the initial product brief for the WidgetWare SDR system:

- business objective;
- intended users;
- inputs and outputs;
- activities the system may perform;
- activities it must not perform;
- approval requirements; and

- measurable acceptance criteria.

1.1 A model is a capability, not a system

A language model predicts and generates language based on its training and current context. It can summarize, classify, compare, draft, infer, and reason. Those abilities are powerful, but they do not define responsibility, permission, persistence, or correctness.

A useful system must surround the model with engineered components. It needs instructions, state, tools, validation, workflows, observability, and tests. When those components are absent, the model may still produce an impressive response, but the response is difficult to trust or integrate.

1.2 Assistants, workflows, agents, and agentic systems

Use precise terms:

- A **model** supplies language and reasoning capability.
- An **assistant** responds to a user within a conversational interface.
- A **workflow** follows a predefined sequence of steps.
- An **agent** selects or adapts actions in pursuit of a goal.
- An **agentic system** combines one or more agents with tools, state, policies, workflows, evaluation, and human oversight.

Not every application needs an agent. A fixed transformation with known rules should normally remain deterministic. Agent behavior is justified when the system must interpret ambiguous input, select among alternatives, synthesize evidence, or adapt its approach.

1.3 The autonomy spectrum

Autonomy should be designed, not assumed.

A practical spectrum is:

1. **Answer only** — the model provides information.
2. **Recommend** — the model proposes an action.
3. **Draft** — the model creates an artifact for review.
4. **Prepare** — the system assembles the data and action parameters.
5. **Execute with approval** — a human authorizes the external action.
6. **Execute within policy** — the system acts autonomously inside narrow limits.

7. **Open-ended autonomy** — the system chooses goals and actions broadly.

WidgetWare will stop at level five. It may research, qualify, and draft, but outbound communication requires human approval.

1.4 Probabilistic reasoning inside deterministic boundaries

The central design pattern is simple:

- Let the model interpret, synthesize, and draft.
- Let software validate, authorize, persist, route, and enforce.

For example, Gemini may judge whether a company fits WidgetWare's ideal customer profile. A deterministic schema validates the result. A workflow decides which step follows. A policy prevents message sending until approval exists.

This is more robust than asking one large prompt to do everything.

1.5 Introducing SDR and WidgetWare

SDR stands for **Sales Development Representative**. An SDR identifies potential customers, researches them, determines whether there is a plausible fit, and begins a conversation that may lead to a sales opportunity.

WidgetWare sells software that helps manufacturing and industrial-automation companies modernize plant operations and adopt AI-enabled automation. Its SDR process requires the system to understand:

- what WidgetWare sells;
- which industries and company characteristics indicate fit;
- what evidence supports a qualification decision;
- what recent events create a relevant reason to contact an account; and
- which statements are approved for outreach.

The system will not scrape indiscriminately, fabricate claims, or contact people without review.

1.6 Initial system boundary

In scope

- accept a target company;

- retrieve approved account information;
- research permitted public evidence;
- evaluate fit against configured criteria;
- produce a structured qualification;
- draft an evidence-backed outreach message; and
- request human approval.

Out of scope

- autonomous prospecting across uncontrolled sources;
- sending email or social messages;
- modifying CRM records without approval;
- inventing customer facts;
- bypassing authentication or source restrictions; and
- making legal, contractual, or pricing commitments.

1.7 Define success before implementation

The first evaluation criteria should be written before code:

- Every qualification result conforms to a schema.
- Every material factual claim includes evidence or is labeled as inference.
- The system does not draft outreach when evidence is insufficient.
- The system never sends a message.
- The system explains the qualification decision.
- The system produces a usable result for representative test accounts.

These are stronger than "the response looks good."

Hands-on lab: Write the system charter

Create:

- `README.md` — project purpose and quick start;
- `SPEC.md` — system behavior and constraints;
- `docs/widgetware-business-brief.md` — products, ICP, and exclusions;
- `docs/acceptance-criteria.md` — measurable success conditions; and
- `tests/scenarios/` — initial scenario descriptions.

No agent code is required yet.

Evaluation checklist

- Is the business objective explicit?
- Are external actions clearly bounded?
- Is human approval mandatory where required?
- Can each acceptance criterion be tested?
- Is the distinction between evidence and inference explicit?
- Does the architecture avoid using an agent where a deterministic rule is sufficient?

Chapter checkpoint

The repository now contains a clear problem definition and system boundary. The reader should be able to explain what the system will do, what it will not do, and what evidence will demonstrate success.

Bridge to Chapter 2

The next chapter builds the engineering harness in Antigravity. Before giving the model more capability, we create the workspace, repository conventions, specifications, permissions, and test discipline that will keep development inspectable.

Exercises

1. Using the autonomy spectrum in §1.3, pick a task you currently automate (or wish you could) and decide honestly which of the seven levels it sits at today, and which level it *should* sit at given how much you currently trust its output. WidgetWare stops at level five; justify in one sentence why your task should stop at the same level, a lower one, or a higher one.
2. Using §1.2's precise vocabulary, describe — without using the word "agent" — what distinguishes an assistant from a workflow, and a workflow from an agent. If you find yourself hedging, that is a sign to revisit this section once Chapter 9 introduces a full multi-agent workflow.
3. WidgetWare's evidence policy is fully specified in Chapter 3 (§3.5) and exercised again in Chapter 8. Before reading either chapter, use §1.6's in-scope and out-of-scope boundary to predict two or three situations where "cite your source" will be harder than it sounds. Revisit this list after Chapter 8 and check how many you anticipated correctly.

4. Using §1.4's central pattern — the model interprets and drafts; software validates, authorizes, persists, routes, and enforces — pick one activity from §1.6's in-scope list and write, in one sentence each, what the model is trusted to decide and what deterministic code must never let it decide alone.
5. Write your own version of §1.7's six success criteria for a WidgetWare capability not yet built — for example, "flag accounts where the qualification score conflicts with the SDR's own instinct." Try to make every criterion something a person could test mechanically, without asking you a clarifying question.
6. Using the Seven Steps to Agent Engineering introduced in this book's Introduction, pick any chapter from Chapter 2 onward before you read it and predict which step it will map to. After reading the chapter's own Seven-Step mapping, check your prediction — where a chapter's *Supporting* steps surprise you, write one sentence on why that step needed revisiting there.

Continue Building the WidgetWare SDR Lab

This Sample Edition ends after Chapter 1. The complete **Book 1 Early Access Edition** continues through ten additional chapters:

- **Chapter 2 — Building with Antigravity:** Establish the specification-driven development harness, repository structure, permissions, and acceptance workflow used throughout the book.
- **Chapter 3 — Gemini Models and Context Engineering:** Select models deliberately and separate stable policy, business context, task context, retrieved evidence, and state.
- **Chapter 4 — Your First Agent with ADK:** Implement the first Google Agent Development Kit agent and inspect sessions, events, and basic state.
- **Chapter 5 — Skills and Reusable Agent Capabilities:** Convert repeatable procedures into versioned, reusable capabilities.
- **Chapter 6 — Structured Outputs and Agent Contracts:** Replace fragile prose handoffs with typed schemas, validation, confidence semantics, and explicit evidence references.
- **Chapter 7 — Tool Engineering:** Build narrow, permission-aware tools with normalized results and testable failure behavior.
- **Chapter 8 — Evidence-Backed Research with MCP:** Research accounts through controlled tools and Model Context Protocol integrations while preserving claim-level evidence.
- **Chapter 9 — Multi-Agent Workflows and Human Approval:** Coordinate specialized agents through explicit state machines, typed handoffs, partial-failure handling, and approval gates.
- **Chapter 10 — Evaluate, Deploy, and Demonstrate:** Create a golden dataset, evaluate the full trajectory, add observability, deploy the system, and define release gates.
- **Chapter 11 — Loop Engineering with ADK:** Wrap the evaluated single-account workflow in a bounded, durable loop that can continue, retry, defer, escalate, or stop.

The complete edition also includes the Book 1 conclusion and a preview of Book 2: **Advanced Architectures**.

Choose Your Next Step

- Register for the free **Community Edition** to receive Chapter 2 in addition to this sample.
- Purchase the complete **Book 1 Early Access Edition** to receive all eleven chapters and future updates to this edition.

Return to the book's Leanpub page to continue.

Thank you for reading.