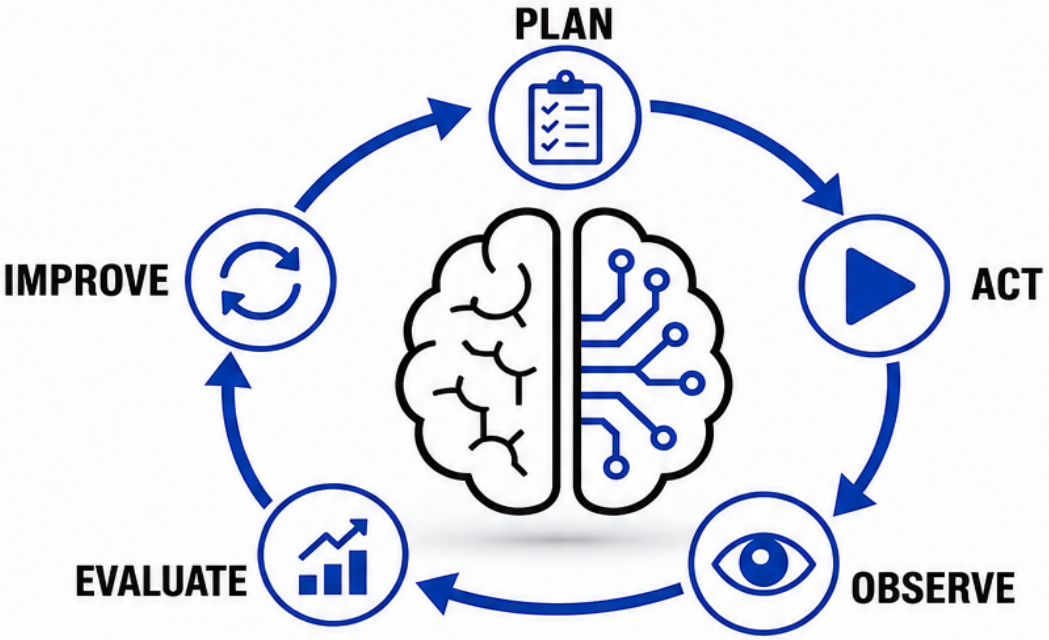
#LOOP ENGINEERING

THE SCIENCE OF AI AGENTS

DESIGNING RELIABLE, SELF-CORRECTING AI SYSTEMS THAT THINK BEFORE THEY ACT



Build reliable, self-correcting autonomous systems using proven loop patterns and leading frameworks including LangGraph, OpenAI Agents SDK, AutoGen, CrewAI, and DSPy.

BRENT TAYLOR

Loop Engineering: The Science of AI Agents

Designing Reliable, Self-Correcting AI Systems That Think Before They Act

Steve Publications

This book is available at
https://leanpub.com/loopengineeringthescienceofaiagents

This version was published on 2026-07-29

Contents

Designing Reliable, Self-Correcting AI Systems That Think Before They Act

This book teaches you how to design, implement, and operate loop-based AI systems from the ground up. You will learn what loop engineering is, why it has become the defining craft of modern AI system design, every major loop pattern used in production agents, and how to build reliable, self-correcting autonomous systems using both custom code and leading frameworks like LangGraph, OpenAI Agents SDK, AutoGen, CrewAI, and DSPy. Whether you are a software engineer starting with AI, an architect designing agent systems, or a practitioner seeking production-grade patterns, this book provides the comprehensive reference you need to move beyond one-shot prompts into the world of agents that reason, act, observe, and improve over time.

Introduction

In June 2026, Boris Cherny, head of Claude Code at Anthropic, said something that has echoed through the AI engineering community: he no longer prompts Claude directly. His job, he said, is to write loops [1]. This was not a throwaway remark. It crystallized a shift that had been building for years and is now the defining divide between prototype AI systems and production-grade autonomous agents.

Consider a simple scenario. You ask a language model to summarize a research paper. The model returns text. If the summary is good, you are done. If it misses key points, you prompt again with corrections, perhaps asking it to focus on methodology or include more technical detail. This back-and-forth works fine for occasional use. It does not scale, and it requires you to be present for every iteration, every correction, every clarification.

Now imagine a system that performs that same iterative refinement without you. The agent reads the paper, produces an initial summary, checks whether it covers all sections, identifies gaps, revises its output, verifies the revision against a rubric, and only then returns the final result. You do not see the intermediate steps unless something goes wrong. The agent handles the iteration, and you get a better outcome with less effort.

That agent is built with loops. Not a metaphorical loop, but actual control structures: while loops, conditional branches, state machines, retry logic, verification gates, and feedback cycles. These are the same primitives you use in every software system you have ever built. Loop engineering is the discipline of applying those primitives to AI systems so that models can think before they act, check their work, learn from mistakes, and pursue goals over extended periods without constant human guidance [2,3].

This book exists because loop engineering has emerged as the central craft of AI system design, yet it remains under-documented. Most tutorials show you how to call an API. Many show you how to chain a few calls together. Fewer still explain how to build systems that iterate intelligently, handle failure gracefully, manage state across dozens of reasoning steps, and operate reliably in production. That gap is what this book fills.

The Promise of This Book

By the time you finish reading, you will be able to:

- Explain what loop engineering is and why it matters, distinguishing it from prompt engineering, workflow orchestration, agent engineering, and traditional software design patterns.
- Design and implement every major loop pattern used in modern AI agents, including planning, reasoning, execution, observation, verification, reflection, self-correction, feedback, memory integration, retrieval-augmented generation, evaluation, optimization, tool use, human-in-the-loop oversight, and multi-agent collaboration.
- Build looped AI systems from scratch using Python and TypeScript, with full control over every aspect of the architecture.
- Use leading frameworks including LangGraph, OpenAI Agents SDK, Microsoft AutoGen (and its successor, the Microsoft Agent Framework), CrewAI, and DSPy, understanding their architectures, trade-offs, and ideal use cases.
- Deploy looped systems in production with proper observability, reliability engineering, cost optimization, security controls, and operational practices.
- Evaluate different architectural choices based on concrete criteria: complexity of the task, required autonomy level, latency constraints, cost budgets, safety requirements, and team expertise.

Who This Book Is For

This book assumes you can write code, preferably in Python, and understand basic concepts like functions, data structures, asynchronous I/O, and API calls. It does not assume prior experience with AI agents or specialized frameworks. If you have used a language model API before, you are ready. If you are a seasoned AI practitioner, you will find detailed coverage of advanced patterns, production considerations, and framework comparisons that go beyond typical blog posts and documentation snippets.

How This Book Is Organized

The book is structured to take you from foundations to mastery in a logical progression. Chapters 1 through 7 establish the conceptual landscape: what loop engineering is, its historical context, the core patterns that form the basis of most agent systems, and how those patterns compose into sophisticated architectures. Chapters 8 through 11 focus on implementation, showing you how to build loops from scratch and then how to use major frameworks, with complete, runnable code examples throughout. Chapters 12 through 14 address production concerns: observability, security, deployment, and operations. Chapter 15 showcases real-world applications across domains, demonstrating how the patterns you have learned apply in practice.

The book does not contain exercises or quizzes. It is designed as a reference you can read sequentially or consult for specific topics, with each chapter written as a complete, self-contained piece of prose that contributes to the whole. Code examples are production-quality: they include setup instructions, dependency management, configuration, testing guidance, and troubleshooting notes so you can adapt them to your own systems.

A Note on the Landscape

The field of AI agents and loop engineering is moving fast. Frameworks evolve, APIs change, and new patterns emerge regularly. This book is written as of mid-2026, drawing on the latest research, documentation, and production practices available at that time. Where frameworks have stable foundations, the guidance will remain relevant for years. Where details are volatile, I note the underlying principles that persist across versions.

Some terminology in this space is still settling. "Loop engineering" itself was popularized in mid-2026 by engineers at Anthropic, Google, and the open-source community [4,5]. Related terms include harness engineering, context engineering, graph engineering, and agent orchestration. I use these terms as the field uses them, noting where they overlap and where distinctions matter.

Now let us begin with the fundamentals: what loop engineering actually is, why it emerged, and the problem it solves.

Chapter 1: What Is Loop Engineering?

The One-Shot Problem

Large language models are remarkable at one-shot tasks. Give them a clear prompt, and they return a useful response. Ask them to write an email, translate text, classify sentiment, or generate a function, and most of the time the output is good enough. This capability has driven enormous adoption across industries and use cases.

One-shot systems also have a fundamental limitation: they succeed only when the model can solve the entire problem in a single pass, using only the information provided in the prompt and its training data. When either assumption fails, the system fails with it.

Consider an AI assistant asked to debug a failing test in a codebase. A one-shot approach would require the model to see the entire relevant codebase, understand the failure, identify the root cause, and produce a correct fix in one response. In practice, this is impossible for non-trivial systems. The codebase may be too large to fit in a context window. The root cause may require running tests, reading logs, or inspecting runtime behavior. The first attempted fix may introduce new issues. None of this can happen in a single pass.

Without loops, the only option is human iteration: the model produces an answer, you evaluate it, refine the prompt, and ask again. This works for occasional use but breaks down at scale. It requires constant human presence, and it does not compound: each interaction starts from scratch, and the system never learns to do better on its own.

Loop engineering solves this problem by building the iteration into the system itself. Instead of relying on humans to iterate, the agent iterates programmatically, using control structures to reason, act, observe results, and adjust its approach until a goal is met or a stopping condition is reached. As we will explore in Chapter 3, this basic idea underlies patterns like ReAct, planning loops, and verification cycles.

A Simple Example That Changes Everything

Let us look at a concrete example that illustrates the difference between one-shot prompting and loop-based design. Imagine building a system that answers questions by searching the web.

A one-shot approach looks like this:

1. User asks a question.
2. System sends the question to an LLM along with instructions to search and answer.
3. The model generates a response, possibly including fabricated information because it cannot actually browse the web in real time.
4. Response is returned to the user.

A loop-based approach works differently:

1. User asks a question.
2. Agent formulates a search query based on the question.
3. Agent calls a search API and retrieves results.
4. Agent reads the results and determines whether they contain sufficient information.
5. If yes, the agent generates an answer grounded in the retrieved content and returns it.
6. If no, the agent formulates a new, more targeted search query and goes back to step 3.
7. The loop continues until either the question is answered or a maximum number of iterations is reached.

This pattern, introduced in the ReAct paper by Yao et al. [6], interleaves reasoning and action in a cycle: the agent thinks about what to do, takes an action, observes the result, and uses that observation to inform the next step. The key insight is that the agent does not need to solve the entire problem upfront. It can discover information incrementally and adapt its strategy as it learns more. Chapter 3 examines this pattern in depth, and Chapter 8 shows how to implement it from scratch in Python.

The difference in outcomes is substantial. One-shot systems hallucinate when they lack information. Loop-based systems can retrieve the missing

information and ground their answers in evidence. One-shot systems fail silently on complex tasks. Loop-based systems can decompose problems, tackle subtasks sequentially, and verify intermediate results along the way.

Defining Loop Engineering

Loop engineering is the practice of designing, implementing, and optimizing the iterative control structures that govern how AI agents reason, act, observe, and adapt over time [7,8]. It encompasses several interconnected concerns:

- **Loop architecture**: The structure of the iteration cycle itself, including what steps are performed, in what order, and under what conditions. Common patterns include observe-think-act cycles, plan-execute-review loops, and generate-evaluate-refine patterns.
- **State management**: How the agent maintains information across iterations, including conversation history, intermediate results, tool outputs, and learned insights. State can be held in memory, persisted to storage, or managed by a framework.
- **Stopping conditions**: The criteria that determine when iteration should end, such as task completion, maximum iteration count, time budget exhaustion, confidence thresholds, or human approval.
- **Fault tolerance**: How the agent handles failures, whether from tool errors, API timeouts, model mistakes, or unexpected inputs. Robust loops include retry logic, fallback strategies, and graceful degradation.
- **Observability**: The ability to trace and understand what the agent is doing at each step, including prompts sent, tools called, decisions made, and time spent. This is essential for debugging non-deterministic systems and ensuring safety.

Loop engineering sits at the intersection of several disciplines. It draws on classical software engineering concepts like control flow, state machines, and error handling. It incorporates AI-specific concerns like prompt design, tool use, and context management. It borrows from operations research and control theory in its treatment of feedback, optimization, and termination conditions.

Why Loops Change the Game

The emergence of loop engineering as a distinct discipline is driven by three factors: the capabilities of modern language models, the demands of production systems, and the limitations of simpler approaches.

Modern models are good enough to iterate. Early language models were too unreliable for autonomous iteration. If an agent made a mistake in step one, it would likely compound that mistake in subsequent steps, leading to increasingly degraded outputs. Modern models, particularly those released from 2023 onward, have sufficient reasoning capability and instruction-following fidelity to perform multi-step tasks with reasonable reliability. This makes loop-based architectures practical where they were previously risky. Chapter 4 explores how verification and self-correction patterns further mitigate the risk of error compounding.

Production demands require more than one-shot responses. In real-world applications, users expect systems that work end-to-end, not prototypes that require constant babysitting. A customer support agent should be able to look up policies, check order history, and draft a response without human intervention for each step. A coding agent should navigate a codebase, make changes, run tests, and fix failures autonomously. These capabilities are only possible with loops. Chapter 15 examines how these patterns play out in production customer support systems and coding agents.

Simpler approaches hit ceilings. Prompt engineering can improve one-shot performance, but it cannot overcome fundamental limitations: if a task requires information the model does not have, or multiple sequential decisions, or verification of intermediate results, no amount of prompt tuning will help. Workflow orchestration provides structure but assumes deterministic paths and fixed sequences. Loop engineering fills the gap by providing structured iteration that adapts to dynamic conditions and model-generated decisions.

Anthropic's engineering team has articulated this clearly: "Agents can handle sophisticated tasks, but their implementation is often straightforward. They are typically just LLMs using tools based on environmental feedback in a loop" [9]. The sophistication comes not from exotic algorithms but from well-designed control structures that let the model leverage its capabilities iteratively. As we will see in Chapters 8 and 9, the same control primitives that make traditional software reliable apply equally well to AI systems.

How This Book Is Organized

The remainder of this book is organized to take you from foundational concepts through advanced implementation techniques and production considerations. Here is the high-level structure:

Foundations (Chapters 2-7): These chapters establish the conceptual landscape. Chapter 2 covers the fundamentals of control flow, state, and termination as applied to AI systems. Chapters 3 and 4 introduce the core loop patterns: planning, reasoning, execution, observation, verification, reflection, and self-correction. Chapters 5 through 7 address memory and retrieval, feedback and optimization, and multi-agent collaboration.

Implementation (Chapters 8-11): These chapters focus on building systems. Chapter 8 shows how to implement loops from scratch in Python, giving you full understanding and control. Chapters 9 and 10 cover leading frameworks: LangGraph for stateful graph-based agents, OpenAI Agents SDK for code-first agent development, and comparisons with AutoGen, CrewAI, and DSPy. Chapter 11 addresses advanced architectural concerns: concurrency, parallelism, event-driven design, and scale.

Production (Chapters 12-15): These chapters deal with deploying and operating systems in the real world. Chapter 12 covers observability, debugging, and reliability engineering. Chapter 13 addresses security, safety, and guardrails. Chapter 14 discusses deployment, CI/CD, monitoring, and cost management. Chapter 15 showcases concrete case studies across domains: coding agents, research agents, customer support systems, document processing pipelines, and self-improving AI systems.

Each chapter is written as a complete piece of prose with detailed explanations, code examples where appropriate, and references to research and documentation. The book is designed to be read sequentially for a comprehensive understanding or consulted selectively for specific topics.

The journey ahead takes you from understanding what loop engineering is to becoming capable of designing and operating production-grade autonomous AI systems. Let us begin with the foundations.

Chapter 2: Foundations - Control Flow for Thinking Machines

From Deterministic Programs to Probabilistic Agents

Traditional software is deterministic. Given the same inputs, a well-written program produces the same outputs every time. This property makes reasoning about programs tractable: you can trace execution paths, predict behavior, and prove correctness using formal methods. Control flow in deterministic programs is explicit and fixed: if statements branch based on conditions you define, loops iterate based on criteria you specify, and functions return values according to logic you write.

AI agents are fundamentally different. At their core lies a language model, which is probabilistic rather than deterministic. Given the same prompt, the model may produce different outputs across calls, especially at non-zero temperature settings. The model does not execute code in the traditional sense; it generates text that you interpret as actions, decisions, or responses. This introduces uncertainty into every step of an agent's execution.

This probabilistic nature changes how we think about control flow. In deterministic programs, you write the logic and trust it to execute correctly. In AI agents, you provide guidance and structure but must account for the possibility that the model will make mistakes, generate unexpected outputs, or fail to follow instructions precisely. Loop engineering is partly the art of building control structures robust enough to handle this uncertainty while still allowing the model's capabilities to shine.

The shift from deterministic to probabilistic execution also changes error handling. In traditional software, errors are exceptional: they indicate bugs, resource failures, or invalid inputs that should be rare. In AI agents, errors are expected: the model may hallucinate, misinterpret instructions, call tools incorrectly, or produce malformed output. Robust loop designs treat these not as exceptions to handle but as normal cases to plan for.

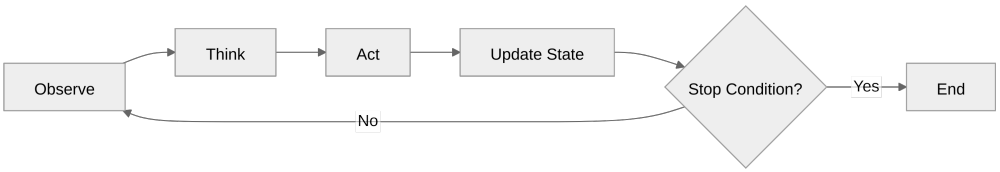
The Anatomy of a Loop

Every loop in an AI agent system, regardless of complexity, shares the same fundamental structure. Understanding this anatomy is essential because it provides a vocabulary for discussing patterns and a mental model for designing new ones.

At its simplest, an agent loop consists of four phases:

1. **Observe**: The agent perceives its current state and environment. This may include reading the user's input, checking tool outputs from the previous step, examining stored memory, or monitoring external systems.
2. **Think**: The agent reasons about what it has observed and decides what to do next. This reasoning is performed by the language model, which generates text that represents its internal deliberation and decision-making process.
3. **Act**: The agent performs an action based on its decision. Actions may include calling a tool, updating memory, sending a message to another agent, or producing output for the user.
4. **Update**: The environment changes in response to the action, and the agent's state is updated to reflect the new situation. This sets up the next observation phase.

These four phases form a cycle that repeats until a stopping condition is met. The cycle is often visualized as:



This simple structure is the foundation for virtually every agent pattern in use today. The ReAct pattern [6] is an explicit instance of this cycle, interleaving reasoning traces with actions. Claude Code's architecture is described as a single-threaded master loop that repeatedly gathers context, takes action, and verifies results [10]. OpenAI's Agents SDK runner performs a tool loop that

switches agents after handoffs and stops when the run finishes or pauses for approval [11].

The power of this pattern lies in its generality. By varying what happens in each phase and how phases are connected, you can implement an enormous range of behaviors: planning systems that decompose tasks before executing them, research agents that iteratively search and synthesize information, coding agents that write code, run tests, and fix failures, or customer support systems that look up policies, check records, and draft responses.

State, Memory, and Context

State is what connects one iteration of a loop to the next. Without state, each iteration would be independent, and the agent could not build on previous work or learn from past experience. In loop engineering, state management is one of the most critical design concerns because it directly affects reliability, cost, and capability.

Agent state typically includes several categories of information:

- **Conversation history**: The sequence of prompts and responses exchanged between the user and the agent. This provides context for understanding the current request in relation to prior interactions.
- **Tool call history**: Records of which tools were called, with what arguments, and what results were returned. This helps the agent avoid redundant actions and track progress toward goals.
- **Intermediate artifacts**: Documents, code snippets, search results, or other data generated during execution that may be needed later.
- **Task state**: Information about the current task, including subtasks completed, remaining work, and any constraints or requirements.
- **Long-term memory**: Persistent knowledge accumulated across sessions, such as user preferences, domain expertise, or lessons learned from past mistakes.

The challenge of state management arises from the limitations of language models. Models have finite context windows: they can only process a limited number of tokens at once. As an agent iterates, the conversation history and tool outputs grow, eventually exceeding the context window. When this

happens, some information must be discarded or compressed, which risks losing critical details needed for correct operation.

Several strategies address this challenge:

- **Selective retention**: Keep only the most relevant parts of the conversation history, discarding older or less important messages. This requires judgment about what is relevant, which can be automated using heuristics or additional model calls.
- **Summarization**: Compress long histories into shorter summaries that preserve key information while reducing token count. Anthropic's guidance for long-running agents emphasizes structured progress logs and session handoffs to bridge context window gaps [12].
- **External memory**: Store information in databases, vector stores, or file systems and retrieve it as needed rather than keeping everything in the context. This is the basis of retrieval-augmented generation (RAG) systems, which we explore in detail in Chapter 5.
- **Structured state schemas**: Define explicit data structures for agent state, separating transient conversational context from persistent task state. Frameworks like LangGraph use typed state objects that flow through graph nodes, making state management explicit and type-safe [13].

The choice among these strategies depends on the application's requirements. Simple agents may need only conversation history in the context window. Complex, long-running agents require sophisticated memory architectures with external storage, retrieval mechanisms, and compression strategies.

Stopping Conditions and Termination

Every loop must eventually stop. An agent that iterates forever consumes resources indefinitely and never delivers a result. Defining appropriate stopping conditions is therefore essential for both correctness and cost control.

Common stopping conditions include:

- **Task completion**: The agent determines, based on its own judgment or an external check, that the task has been completed successfully. This is the ideal condition but requires the agent to have reliable self-assessment capability.

- **Maximum iterations**: A hard limit on the number of loop iterations. This prevents infinite loops but may cut off execution before the task is complete if the limit is too low.
- **Time budget**: A maximum duration for execution, measured in wall-clock time or compute units. Useful for real-time systems where latency matters more than completeness.
- **Token budget**: A maximum number of tokens consumed across all model calls. Directly controls cost but requires careful monitoring and may terminate mid-operation.
- **Confidence threshold**: The agent stops when its self-reported confidence in the output exceeds a specified level. Relies on the agent's ability to calibrate confidence, which is not always reliable.
- **External verification**: An independent check, performed by another model, a rule-based system, or a human, confirms that the output meets quality standards.
- **User intervention**: A human explicitly approves, rejects, or modifies the agent's work, ending the loop.

Production systems typically combine multiple stopping conditions for robustness. A coding agent might stop when tests pass (task completion), but also enforce a maximum iteration count and token budget to prevent runaway execution. A customer support agent might stop when it has drafted a response and verified it against policies, but require human approval before sending.

The design of stopping conditions involves trade-offs between reliability, cost, and latency. Stricter conditions improve reliability but increase resource consumption. Looser conditions reduce cost and latency but risk incomplete or incorrect outputs. The right balance depends on the application's requirements and risk tolerance.

A critical principle in loop engineering is that stopping conditions should be enforced by the system, not trusted to the model. Language models are unreliable at self-termination: they may claim a task is complete when it is not, or they may fail to recognize completion and continue unnecessarily. Well-designed systems implement explicit checks outside the model's control to enforce termination policies. Chapter 13 discusses safe stopping conditions in detail, including budget enforcement and kill switches.

Iteration Versus Recursion in Agent Design

Control flow in agent systems can be structured using iteration (loops that repeat) or recursion (functions that call themselves). Both approaches are valid and appear in production systems, but they have different characteristics and trade-offs.

Iteration is the simpler and more common approach. An iterative agent runs a loop that repeats until a stopping condition is met. Each iteration updates the agent's state and moves closer to completion. Iteration is easy to reason about, straightforward to implement, and naturally bounded by iteration limits or time budgets. Most basic agent patterns are iterative: ReAct loops, tool-calling agents, and simple plan-execute-review cycles all use iteration.

Recursion appears in more sophisticated architectures where tasks are decomposed into subtasks that may themselves require further decomposition. A recursive agent receives a task, determines whether it can handle the task directly or needs to break it down, and if needed, spawns sub-agents (or calls itself) to handle subtasks. Results from subtasks are combined to produce the final output.

Recursive architectures are powerful for complex, hierarchical problems but introduce additional complexity:

- **Termination**: Recursive agents must have clear base cases that prevent infinite recursion. Each level of recursion should make progress toward a simpler problem.
- **Resource management**: Each recursive call consumes tokens and compute. Without careful control, deep recursion can exhaust budgets quickly.
- **State coordination**: Results from recursive calls must be aggregated and integrated into the parent task's state. This requires careful design of return values and communication protocols.
- **Debugging**: Tracing execution through multiple levels of recursion is more difficult than following a linear or iterative flow.

In practice, many systems combine iteration and recursion. A top-level agent might iteratively execute steps in a plan, but some steps involve recursive

decomposition into subtasks handled by specialized sub-agents. LangGraph's state graph model supports both patterns: cycles in the graph represent iteration, while nested graphs represent recursion [13].

The choice between iteration and recursion depends on the problem structure. Simple tasks that require repeated refinement are well-suited to iteration. Complex tasks with natural hierarchical decomposition may benefit from recursion. Many production systems use iteration at the core loop level and recursion for task decomposition, getting the benefits of both approaches.

Understanding these foundations - control flow, state management, stopping conditions, and structural patterns - provides the conceptual toolkit needed to design effective agent loops. The next chapters build on this foundation by introducing specific loop patterns and showing how they are implemented in practice.

Chapter 3: Core Loop Patterns - Planning, Reasoning, Execution, Observation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/loopengineeringthescienceofaiagents.

The ReAct Pattern

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/loopengineeringthescienceofaiagents.

Planning Loops

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/loopengineeringthescienceofaiagents.

Reasoning Loops (Chain-of-Thought and Beyond)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/loopengineeringthescienceofaiagents.

Execution Loops and Tool Use

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/loopengineeringthescienceofaiagents.

Observation Loops and Perception

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/loopengineeringthescienceofaiagents.

Composing Core Patterns

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/loopengineeringthescienceofaiagents.

Chapter 4: Self-Correction - Verification, Validation, Reflection, Critique

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/loopengineeringthescienceofaiagents.

Verification Loops

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/loopengineeringthescienceofaiagents.

Validation Loops

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/loopengineeringthescienceofaiagents.

Reflection Loops

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/loopengineeringthescienceofaiagents.

Critique Loops

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/loopengineeringthescienceofaiagents.

Self-Correction and Retry Loops

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/loopengineeringthescienceofaiagents.

Combining Correction Patterns

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/loopengineeringthescienceofaiagents.

Chapter 5: Memory, Retrieval, and Knowledge Integration

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/loopengineeringthescienceofaiagents.

The RAG Loop

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/loopengineeringthescienceofaiagents.

Short-Term Versus Long-Term Memory

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/loopengineeringthescienceofaiagents.

Memory Update Loops

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/loopengineeringthescienceofaiagents.

Context Window Management

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/loopengineeringthescienceofaiagents.

Knowledge Consolidation and Forgetting

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/loopengineeringthescienceofaiagents.

Chapter 6: Feedback, Evaluation, and Optimization Loops

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/loopengineeringthescienceofaiagents.

Human-in-the-Loop Patterns

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/loopengineeringthescienceofaiagents.

Automated Evaluation Loops

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/loopengineeringthescienceofaiagents.

Preference Learning and RLHF-Style Loops

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/loopengineeringthescienceofaiagents.

DSPy-Style Optimization Loops

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/loopengineeringthescienceofaiagents.

Online Versus Offline Learning Loops

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/loopengineeringthescienceofaiagents.

Chapter 7: Multi-Agent Collaboration and Coordination

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/loopengineeringthescienceofaiagents.

Why Multiple Agents?

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/loopengineeringthescienceofaiagents.

Conversation and Debate Loops

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/loopengineeringthescienceofaiagents.

Hierarchical and Managerial Loops

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/loopengineeringthescienceofaiagents.

Swarm and Emergent Behavior

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/loopengineeringthescienceofaiagents.

Conflict Resolution and Consensus

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/loopengineeringthescienceofaiagents.

Chapter 8: Building Loops from Scratch in Python

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/loopengineeringthescienceofaiagents.

Project Structure and Dependencies

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/loopengineeringthescienceofaiagents.

A Minimal ReAct Agent

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/loopengineeringthescienceofaiagents.

State Management Without Frameworks

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/loopengineeringthescienceofaiagents.

Tool Calling Infrastructure

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/loopengineeringthescienceofaiagents.

Error Handling and Fault Tolerance

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/loopengineeringthescienceofaiagents.

Testing Your Custom Loops

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/loopengineeringthescienceofaiagents.

TypeScript Implementation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/loopengineeringthescienceofaiagents.

Chapter 9: LangGraph - Production-Grade Cyclic Agents

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/loopengineeringthescienceofaiagents.

The LangGraph Mental Model

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/loopengineeringthescienceofaiagents.

Building a Simple State Graph

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/loopengineeringthescienceofaiagents.

Conditional Edges and Branching Logic

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/loopengineeringthescienceofaiagents.

Human-in-the-Loop with Interrupts

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/loopengineeringthescienceofaiagents.

Persistence and Checkpointing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/loopengineeringthescienceofaiagents.

Production Patterns in LangGraph

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/loopengineeringthescienceofaiagents.

Complete LangGraph TypeScript Project

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/loopengineeringthescienceofaiagents.

Chapter 10: Framework Landscape - LangChain, OpenAI Agents, AutoGen, CrewAI, DSPy

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/loopengineeringthescienceofaiagents.

LangChain and LCEL

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/loopengineeringthescienceofaiagents.

OpenAI Agents SDK

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/loopengineeringthescienceofaiagents.

Microsoft AutoGen

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/loopengineeringthescienceofaiagents.

CrewAI

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/loopengineeringthescienceofaiagents.

DSPy

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/loopengineeringthescienceofaiagents.

Choosing the Right Tool

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/loopengineeringthescienceofaiagents.

Chapter 11: Advanced Architecture - Concurrency, Events, and Scale

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/loopengineeringthescienceofaiagents.

Asynchronous Execution Patterns

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/loopengineeringthescienceofaiagents.

Parallelism in Agent Loops

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/loopengineeringthescienceofaiagents.

Event-Driven Loop Architectures

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/loopengineeringthescienceofaiagents.

Distributed and Multi-Process Agents

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/loopengineeringthescienceofaiagents.

Latency and Cost Optimization

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/loopengineeringthescienceofaiagents.

Chapter 12: Observability, Debugging, and Reliability

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/loopengineeringthescienceofaiagents.

Tracing and Logging Loops

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/loopengineeringthescienceofaiagents.

Debugging Non-Deterministic Behavior

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/loopengineeringthescienceofaiagents.

Fault Tolerance Patterns

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/loopengineeringthescienceofaiagents.

Benchmarking and Performance Analysis

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/loopengineeringthescienceofaiagents.

Alerting and Incident Response

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/loopengineeringthescienceofaiagents.

Chapter 13: Security, Safety, and Guardrails

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/loopengineeringthescienceofaiagents.

Attack Surfaces in Looped Agents

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/loopengineeringthescienceofaiagents.

Prompt Injection and Jailbreak Resilience

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/loopengineeringthescienceofaiagents.

Tool Call Validation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/loopengineeringthescienceofaiagents.

Safe Stopping Conditions

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/loopengineeringthescienceofaiagents.

Guardrail Patterns

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/loopengineeringthescienceofaiagents.

Chapter 14: Production Deployment and Operations

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/loopengineeringthescienceofaiagents.

Containerization and Orchestration

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/loopengineeringthescienceofaiagents.

CI/CD for Agent Systems

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/loopengineeringthescienceofaiagents.

Monitoring in Production

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/loopengineeringthescienceofaiagents.

Cost Management at Scale

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/loopengineeringthescienceofaiagents.

Chapter 15: Real-World Applications and Case Studies

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/loopengineeringthescienceofaiagents.

Coding Agents and Software Engineering

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/loopengineeringthescienceofaiagents.

Research Agents

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/loopengineeringthescienceofaiagents.

Customer Support Systems

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/loopengineeringthescienceofaiagents.

Document Processing Pipelines

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/loopengineeringthescienceofaiagents.

Self-Improving AI Systems

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/loopengineeringthescienceofaiagents.

Deep Dive: SWE-agent Architecture and Performance

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/loopengineeringthescienceofaiagents.

Deep Dive: Production Customer Support at Scale

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/loopengineeringthescienceofaiagents.

Production Failure Case Studies

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/loopengineeringthescienceofaiagents.

Emerging Patterns and Future Directions

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/loopengineeringthescienceofaiagents.

Conclusion

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/loopengineeringthescienceofaiagents.

Key Lessons

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/loopengineeringthescienceofaiagents.

The Future of Loop Engineering

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/loopengineeringthescienceofaiagents.

Final Thoughts

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/loopengineeringthescienceofaiagents.

Glossary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/loopengineeringthescienceofaiagents.

References

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/loopengineeringthescienceofaiagents.