The Agentic Development Loop: A New Paradigm for Software Delivery

By Andy Slack | February 23, 2026 | AI Development

Stop relying on linear AI coding. The Agentic Development Loop uses self-correcting agents to fix bugs and build features. Read our guide on implementing CodeRabbit, TDD, and AI guardrails.

The concept of an Agentic Loop is fundamentally transforming the way we approach software development. Moving beyond the traditional "one-shot" linear workflow, an agentic loop is an iterative, self-correcting process that continues until a pre-defined success metric is achieved.

When we talk about "Agentic Loop Development," we're essentially delegating a full development task-like "build feature X" or "fix bug Y"-to an intelligent agent. The agent is responsible for taking the initial requirements and iterating until the task is successfully completed without human intervention.

This looping methodology is crucial because, in complex modern software systems, the old "one-shot" approach frequently leads to integration issues or breaking changes. As system complexity increases, the likelihood of a linear change causing unforeseen problems escalates dramatically.

How the Agentic Development Loop Works

While specific implementations may vary, our Agentic Development Loop follows three core phases:

- Code Creation: Based on the initial input (a development task or bug report), the agent creates the necessary code. We adhere to trunk-based development, ensuring the agent branches off to work on a feature branch, maintaining a clean main codebase.
- Linting, Building, and Testing: Just like any professional developer, the agent must perform quality checks before submitting its work for review. This phase ensures the code meets all established standards:
- Passes all linting checks.
- Builds without error.
- Passes all automated tests.
- Code Review and Iteration: The agent's work is then reviewed, ideally by a separate model to provide an objective, second set of eyes, mimicking the standard human developer workflow.
- Success: If no issues are found, the loop is successful. The code can be processed (e.g., opening a Pull Request or confident merging).
- Iteration: If the review finds areas for improvement, these feedback points become the new, refined input prompts for the agent to start another loop.

Measuring Success: Speed and Quality

In software development, success often lies in balancing speed and quality. For established teams, this is often measured by DORA metrics. For smaller organisations, it's a more general sense of fast delivery with minimal issues.

By adopting an agentic loop development methodology, you gain a key benefit in both areas:

- Speed: Agents can work on an unlimited number of tasks simultaneously, operating 24/7. Historically, coding was the bottleneck; this is no longer the case. You are more likely to find your limit is in generating work and performing UAT (User Acceptance Testing) as the throughput volume increases.
- Quality: Quality is highly dependent on how well your projects are configured. Our goal is to ensure the loop consistently builds clean code (no "slop" or security issues) that meets all user requirements and operates without human developer interference.

The Ideal Agentic Setup

To ensure you achieve maximum quality, we recommend the following critical implementations:

- Aggressive Code Review: Install a tool like CodeRabbit (https://www.coderabbit.ai/) into your repository and set it to ASSERTIVE mode. This ensures your code review agent has robust, challenging feedback to work with.
- Clear Agent Instructions: Maintain an AGENTS.MD (http://AGENTS.MD) file in your repository. This file must provide clear, explicit instructions on how agents should operate within your specific environment, tooling, and conventions.
- Test-Driven Development (TDD): Instruct your agents to work with a TDD approach. They must write tests for every feature or fix. The ultimate goal is full test coverage across the board.
- Use front-end testing tools like Playwright for web applications.
- Implement both unit and e2e (end-to-end) tests for backend applications.
- DRY Principles: Enforce DRY (Don't Repeat Yourself) principles, requiring agents to abstract duplicate code into components or functions to prevent code bloat and maintain a lean codebase.
- Strict Standards: Enforce strong linting practices to maintain high, consistent code standards.
- Automated Formatting: Use a strong, consistent formatting system (e.g., Prettier) to ensure all code is styled in a way your human team is comfortable reviewing and maintaining.

The Evolving Role of the Developer

It is easy to get carried away and believe agents will eliminate the need for human developers, but our view is that the work doesn't go away-it changes.

If your current developers aren't utilising code-writing agents, you'll be missing massive productivity gains. However, if all your code is written by agents, a new set of critical roles emerges:

- Prompt Engineers & Refiners: Who is evolving the agent prompts to align with your changing business needs and improve performance?
- Guardrails & Strategy Auditors: Who is ensuring the overall testing strategies, quality checks, and security guardrails are effective and working?

- Business-to-Technical Translators: Most importantly, who is translating ambiguous business requirements into precise, technical specifications that the agents need to deliver correct results with limited back-and-forth?

Think of it like a tube of toothpaste: Squeezing the middle (the coding phase) doesn't make the paste vanish; it just shifts the pressure.

AI accelerates coding, but this pressure is immediately transferred, making Specifications and Testing the new, critical bottlenecks. The job remains the same: translating business goals into technical reality and ensuring the resulting product works.

The Result for Your Business

You have a critical choice: adopt agentic development or partner with companies that do. Your business has the potential to move 10x or even 100x faster, enabling you to keep pace with-or even outpace-your competition. To ignore these early warning signs is to fall behind, a gap that is already accelerating.

The challenge is that as technology evolves, the void between those who adopt agentic strategies and those who do not will become so vast that it may become impossible to bridge.

At JuicyLlama, we specialize in building the internal tooling and strategic frameworks that enable our clients to move at this rapid pace, creating strong technical barriers to entry for their competitors. If you're interested in learning more, get in touch.

Further Reading:

Building Effective Agents (Anthropic) Why read it: This guide serves as the technical blueprint for the "Agentic Loop" concept. Anthropic argues against using heavy frameworks, instead favoring "composable patterns" to ensure reliability. It details five specific architectures-such as Orchestrator-Workers and Evaluator-Optimizer-that provide the actual engineering structure needed to build self-correcting, iterative software agents. Read the engineering guide (https://www.anthropic.com/engineering/building-effective-agents)

Andrew Ng's "Agentic Design Patterns" (DeepLearning.AI (http://DeepLearning.AI)) Why read it: Andrew Ng has been a massive proponent of the idea that "Agentic Workflows" (iterative loops) drive better results than better models. He breaks down patterns like Reflection (checking your own work) and Tool Use in a way that perfectly complements the Anthropic article. Read the summary on The Batch (https://www.deeplearning.ai/the-batch/issue-242/)

LLM Powered Autonomous Agents (Lilian Weng, OpenAI) Why read it: This is widely considered the "Bible" of AI Agent theory. It breaks down the agent into three components: Planning (the loop), Memory, and Tool Use. It provides the theoretical backbone for what Anthropic discusses practically. Read it here (https://lilianweng.github.io/posts/2023-06-23-agent/)

SWE-bench: Can Language Models Resolve Real-World GitHub Issues? Why read it: Anthropic explicitly mentioned using SWE-bench to test their agents. Reading the methodology of this benchmark will give you deep insight into how "Coding Agents" are measured and where they typically fail (integration issues, wrong file paths, etc.). Visit the project (https://www.swebench.com/)

Continue the conversation: