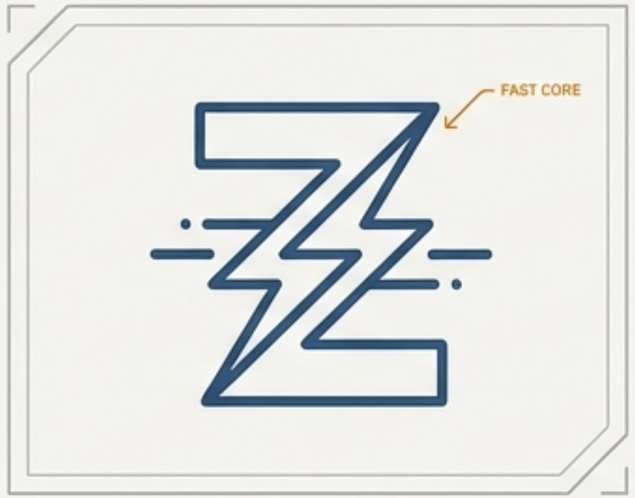
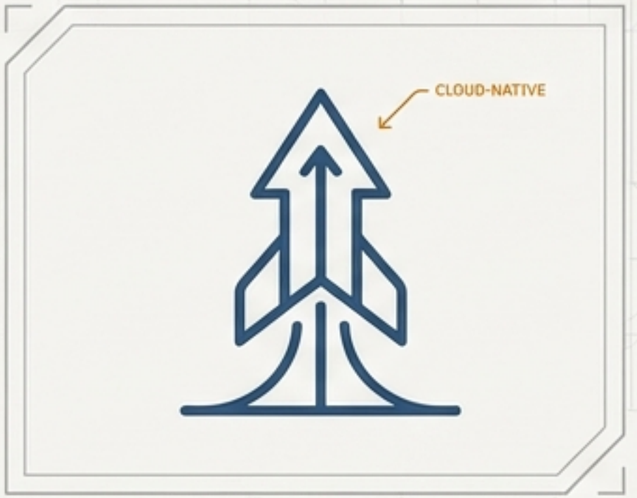
The New Stack: A Developer's Guide to AI-Native IDEs

Comparing Zed, Cursor, and Antigravity in a Head-to-Head Challenge





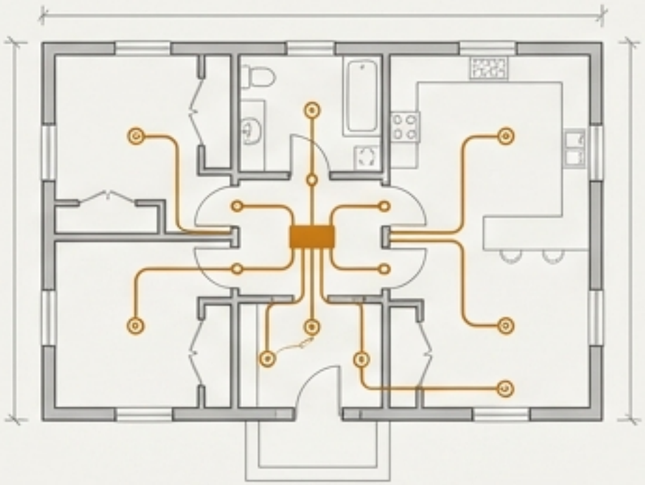
The Paradigm Shift from “AI Features” to “AI-Native”

An **AI-native IDE** is a code editor designed from the ground up **with AI collaboration** as a central architectural principle, not an afterthought.

Traditional Editor + AI Plugin

Like adding electricity to an old house. The wiring has to work around existing walls.

AI-Native Editor



Like building a modern house. The wiring is planned before pouring the foundation.

Context-Aware AI: Understands your entire project, not just a single file.

Multi-Model Support: Use different AI models for different tasks (e.g., Claude for quality, GPT-4 Mini for speed).

Agent Capabilities: AI can work autonomously on tasks spanning multiple files.

Meet the Contenders

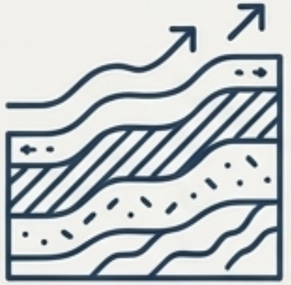
Zed

The Speedster

Built from scratch in Rust for maximum performance. Designed for direct, inline collaboration with instant AI responses and a startup time under 200ms.

- **Best For:**
  Solo developers prioritizing speed and a clean, minimalist interface.

Cursor



The Evolution

A fork of VS Code, rebuilt with deep AI integration. Combines a familiar interface and extension support with powerful agentic capabilities.

- **Best For:**
  VS Code users wanting AI-native features without leaving a mature ecosystem.

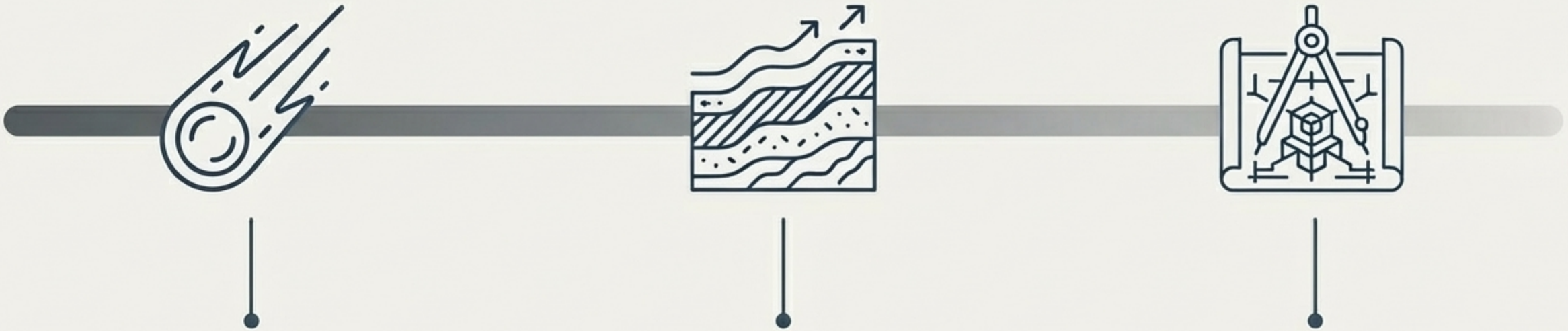
Antigravity

The Architect

An agent-first control plane that emphasizes planning, artifacts, and parallel work. You approve the blueprint before the build.

- **Best For:**
  Complex, agent-driven workflows and projects requiring significant architectural planning.

The Spectrum of Collaboration: From Tool to Teammate



Direct Control

You are the developer.
The AI is your hyper-competent assistant.

Delegation & Review

You are the manager.
The AI is your autonomous engineer.

Strategic Oversight

You are the architect.
The AI is your engineering team.

The Arena: A Standardized Challenge

The Task Manager CLI

```
$ task add "Deploy to production"
Task #3 added.

$ task list
ID | Description          | Status    | Created
-----------------------------------------------
1  | Refactor auth module | Pending   | 2025-11-20
2  | Write documentation  | Completed | 2025-11-19
3  | Deploy to production | Pending   | 2025-11-21
```

Core Requirements

- `add <task>`: Add a new task with a unique ID and timestamp.
- `list`: Display all tasks in a formatted table.
- `complete <id>`: Mark a task as completed.
- `delete <id>`: Remove a task from the list.

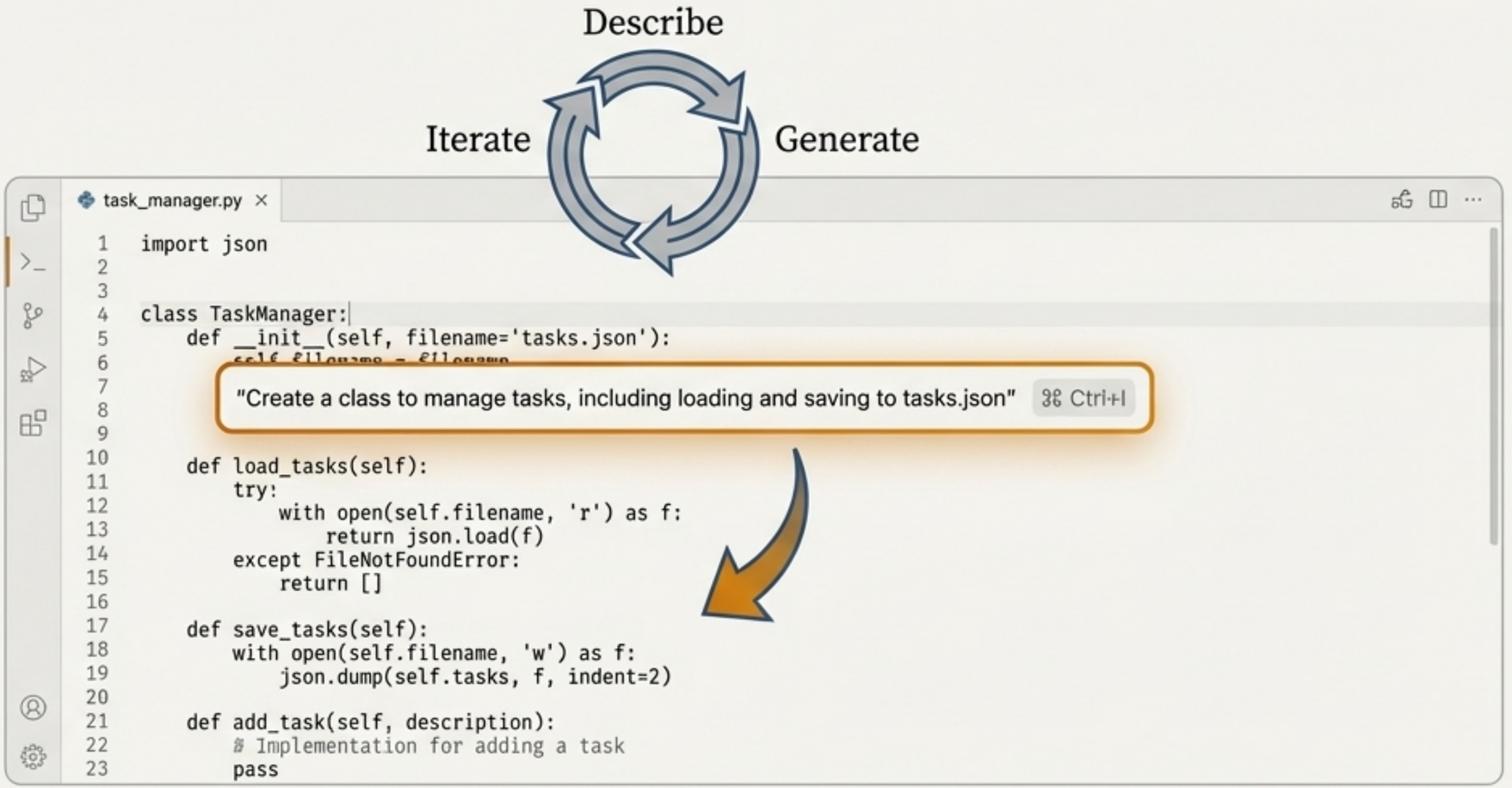
Data & Persistence

Tasks are stored in a `tasks.json` file. Each task includes: `id`, `description`, `priority`, `completed`, `created_at`.

Why This Project is a Good Test

It requires state management and file I/O.
It involves iterative feature development.
It tests code generation, context awareness, and refactoring.

Workflow 1: Zed's Inline Velocity

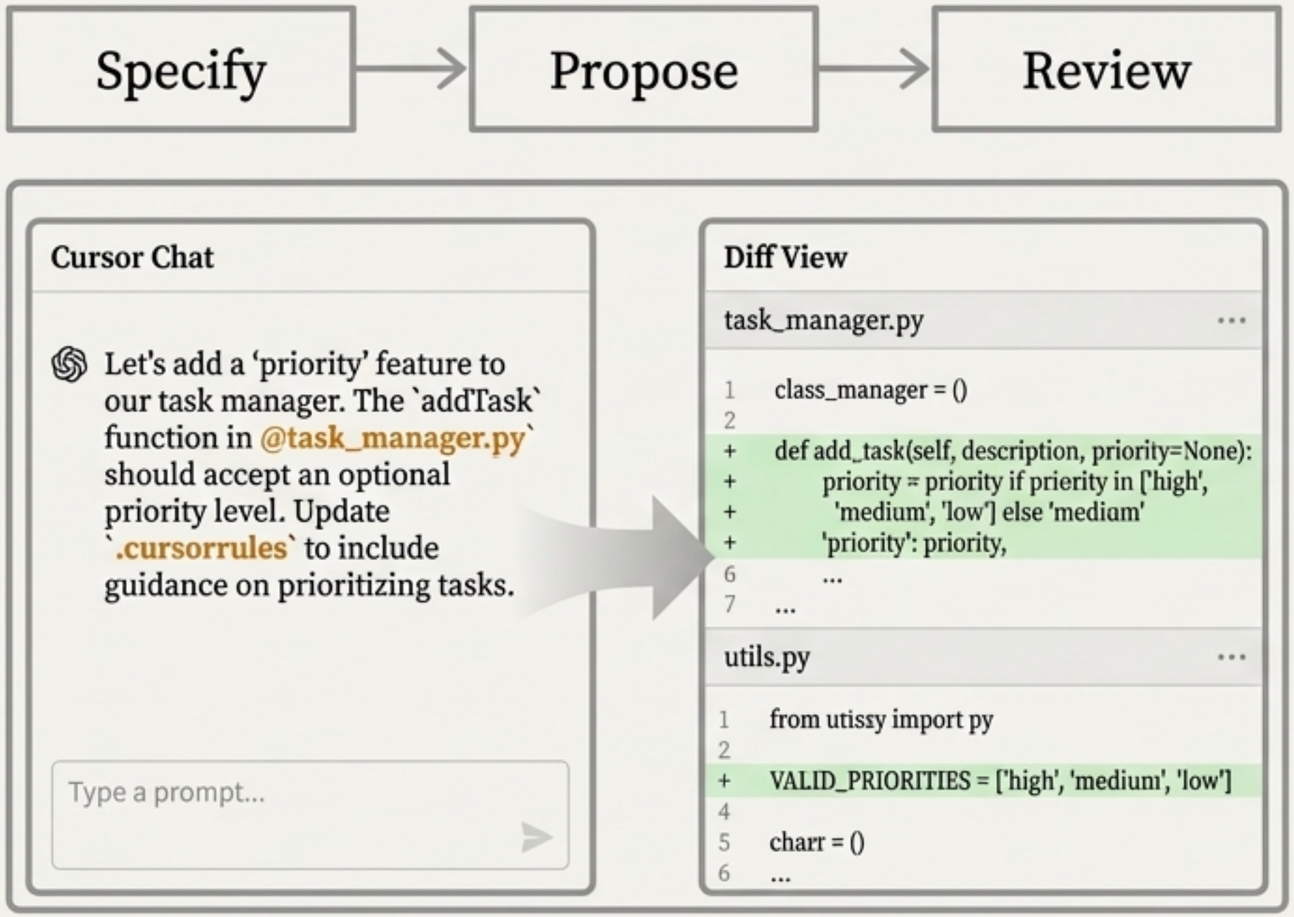


Key Interaction Pattern

1. **Describe Intent:** You write a comment or position the cursor and press `Ctrl+I`.
2. **Generate Code:** The AI generates code instantly, directly in the editor.
3. **Iterate Rapidly:** You refine the result with another `Ctrl+I` prompt, like "Add error handling for invalid task IDs".

"Building function by function with rapid feedback. The AI is embedded in your editing environment, not separate from it."

Workflow 2: Cursor's Agentic Approach

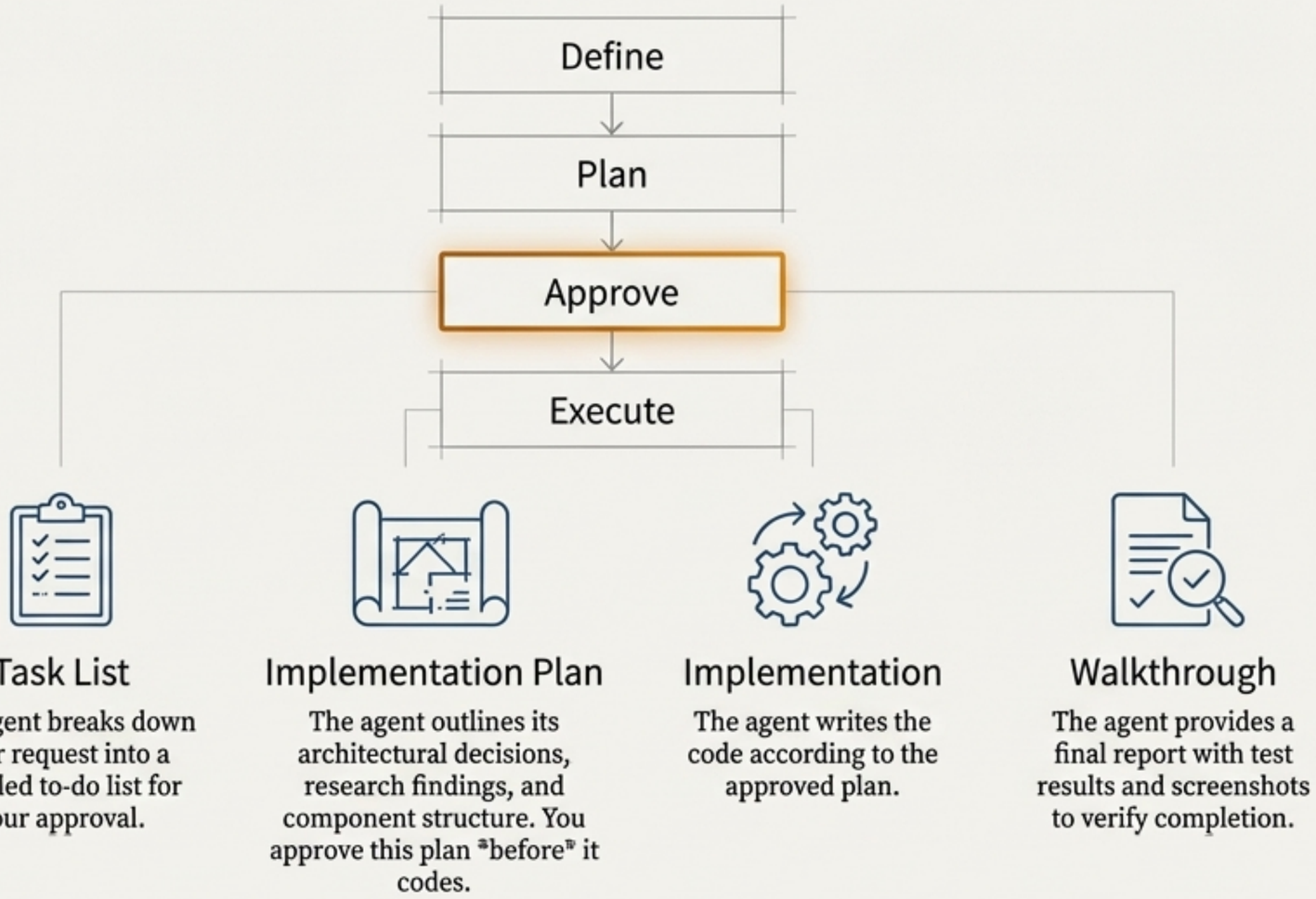


Key Interaction Pattern

1. **Specify Upfront:** You provide a detailed specification in the Chat pane, often referencing multiple files (`@file`) and project-wide rules (`.cursorrules`).
2. **Propose Solution:** The AI agent analyzes the spec and proposes a complete, multi-file implementation.
3. **Review Diff:** You review the proposed changes in a diff editor, accepting, rejecting, or modifying them before they are applied.

"Planning upfront for a comprehensive first draft. You act as a reviewer, approving changes proposed by your autonomous engineer."

Workflow 3: Antigravity's Architectural Workflow

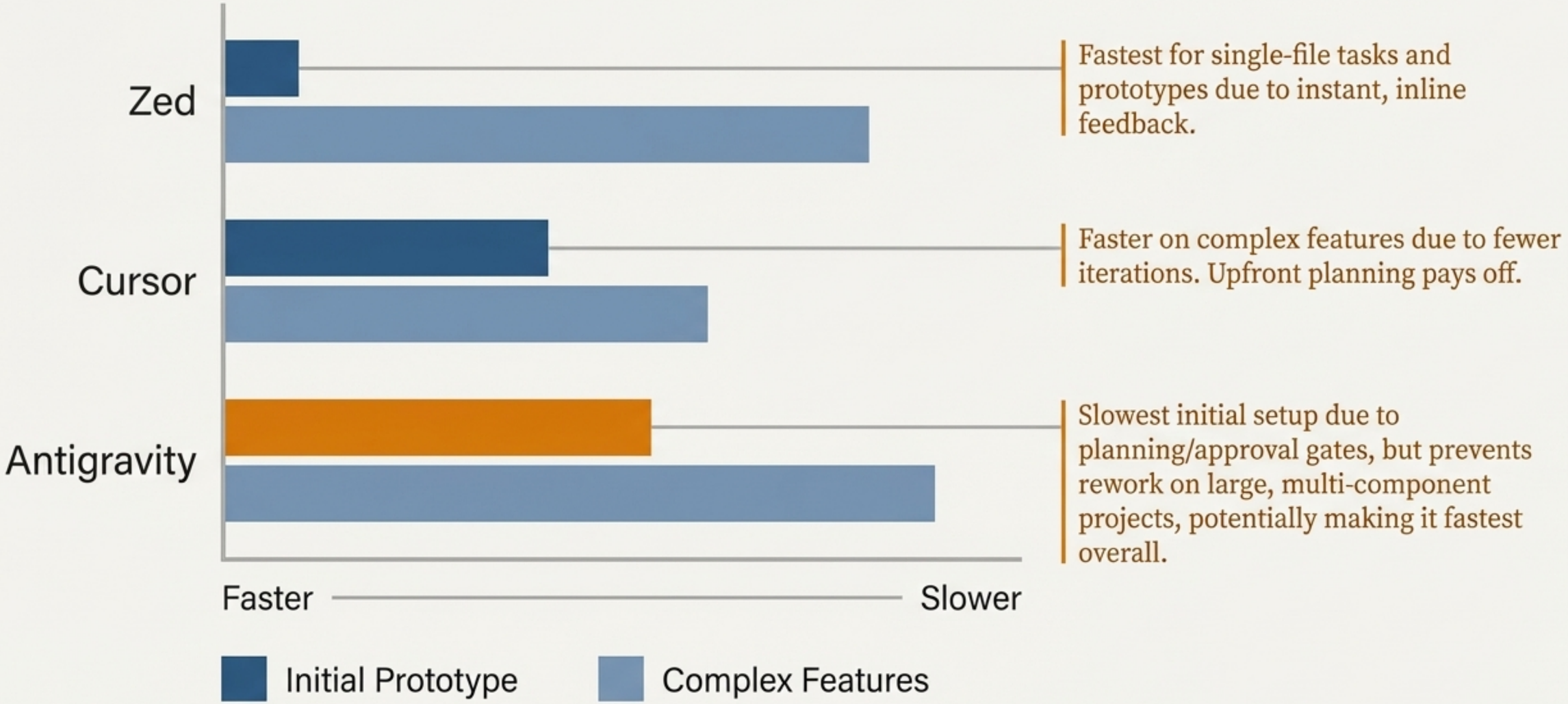


Key Interaction Pattern

1. **Define Task**
   You give the agent a high-level goal.
2. **Review & Approve Plan**
   The agent generates artifacts (Task List, Implementation Plan). You review and approve its strategy.
3. **Monitor Execution**
   The agent implements the approved plan, providing updates.

"Reviewing the blueprint before the build. This catches architectural issues early and ensures the final code aligns with a pre-approved plan."

Head-to-Head: Development Speed & Velocity

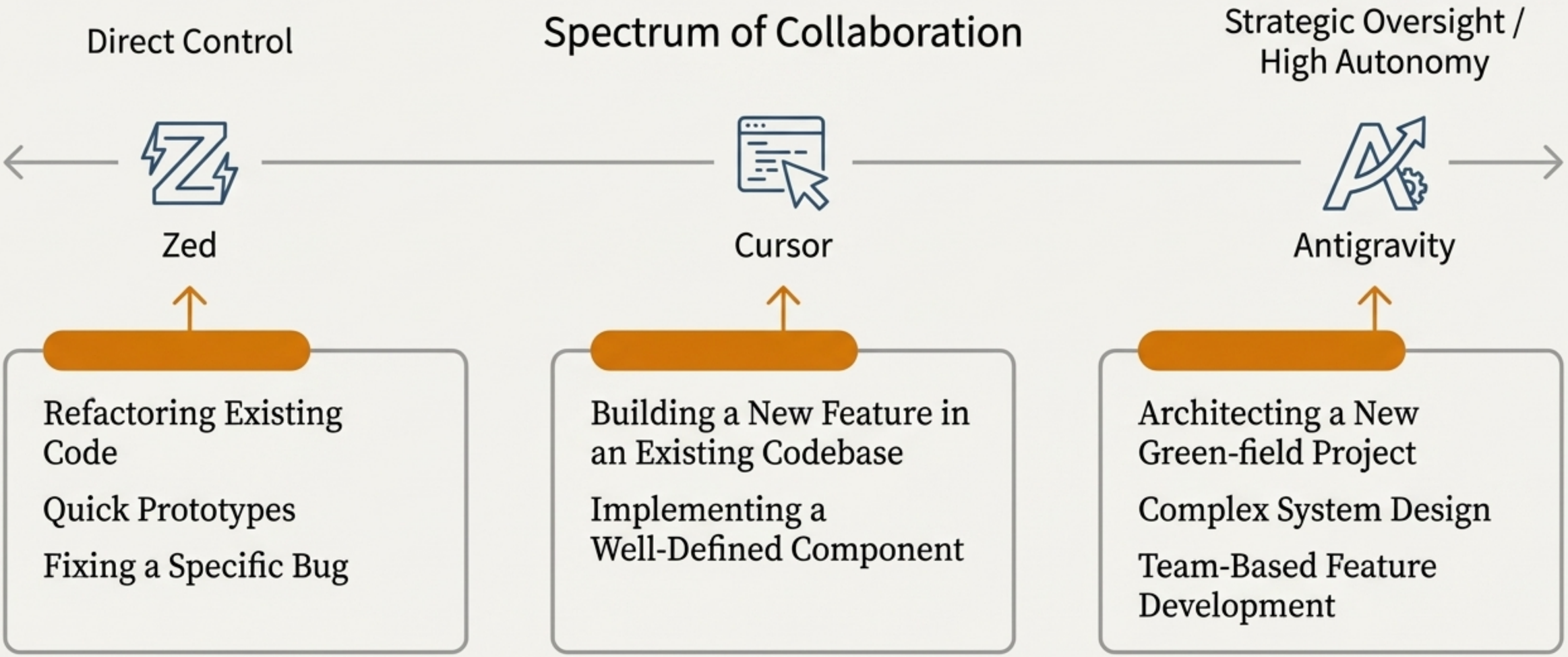


Fastest for single-file tasks and prototypes due to instant, inline feedback.

Faster on complex features due to fewer iterations. Upfront planning pays off.

Slowest initial setup due to planning/approval gates, but prevents rework on large, multi-component projects, potentially making it fastest overall.

Head-to-Head: Autonomy vs. Control



Head-to-Head: Code Quality & Character

Zed

```
def calculate_sum(numbers):
    return sum(numbers)

data = [1, 2, 3, 4, 5]
result = calculate_sum(data)
print(result)
```

The code is correct, efficient, and directly responsive to the prompt. It's clean and unopinionated, without extra boilerplate.

Character
Like a skilled pair programmer delivering exactly what you asked for, right now.

Cursor

```
from typing import List, Optional

def calculate_average(numbers: List[float]) -> Optional[float]:
    """Calculates the average of a list of numbers.
    Args:
        numbers: A list of floating-point numbers.
    Returns:
        The average of the numbers, or None if the list is empty.
    """
    if not numbers:
        return None
    try:
        total = sum(numbers)
        return total / len(numbers)
    except TypeError as e:
        print(f"Error: {e}")
        return None

data = [10.0, 20.0, 30.0]
avg = calculate_average(data)
print(avg)
```

The code is more comprehensive, often including defensive checks, detailed docstrings, and consideration for edge cases not explicitly mentioned in the prompt.

Character
Like a senior engineer who anticipates future problems and builds for maintainability.

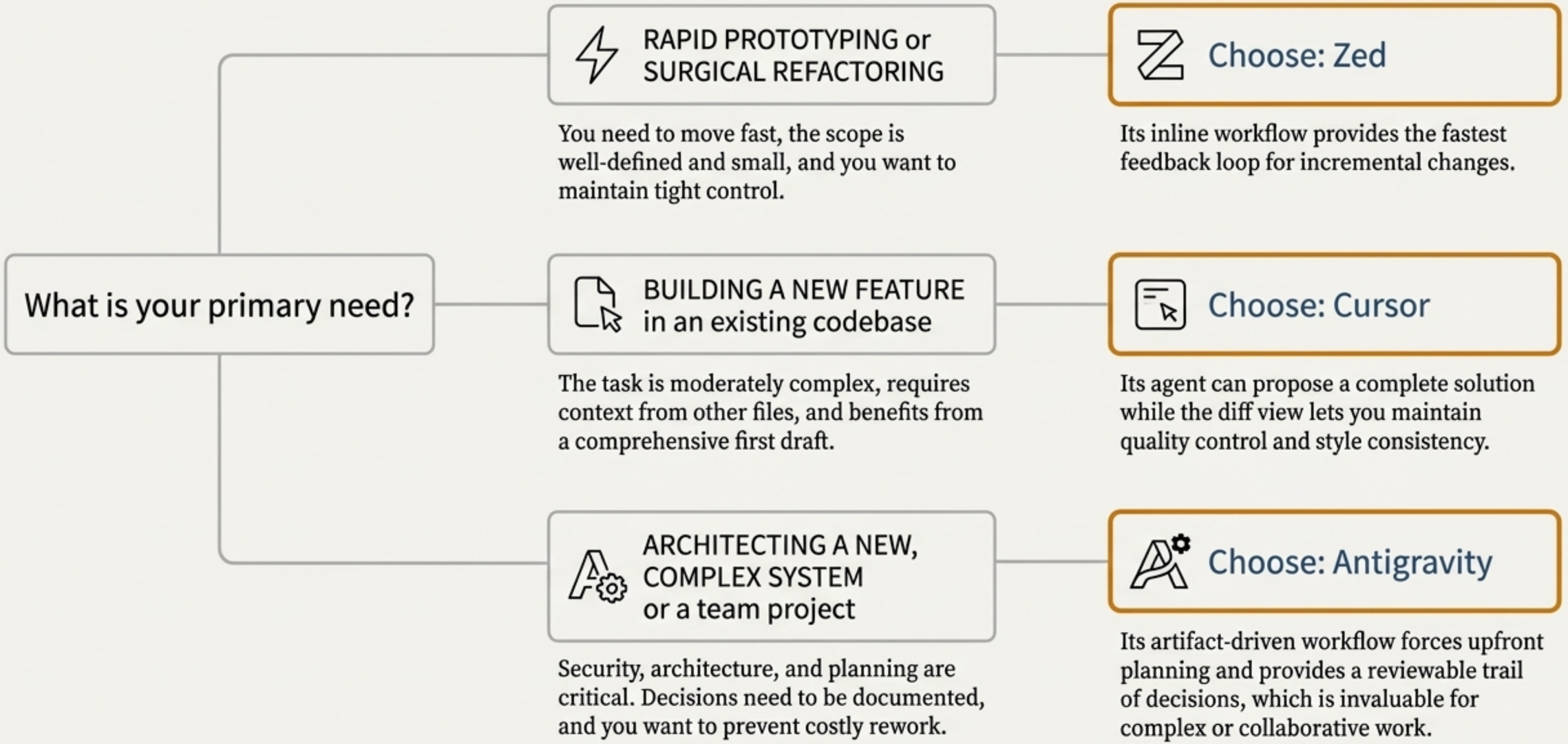
Antigravity

```
# As per Plan 2.1, using a class-based approach for modularity.

class DataProcessor:
    def __init__(self, data_source: str):
        self.data_source = data_source
        # Plan 2.2: Initialize logging with context
        self.logger = self._setup_logger()

    def _setup_logger(self):
        # Implementation details for logger setup
        pass

    def process(self):
        # Plan 2.3: Main processing loop
        # Steps aligned with system design doc
        print(f"Processing data from {self.data_source}...")
        # Further modular components here

if __name__ == "__main__":
    processor = DataProcessor("input_data.csv")
    processor.process()
```

The code is highly structured and clearly aligned with the pre-approved Implementation Plan. It's often more modular and includes comments explaining *why* certain architectural decisions were made.

Character
Like an engineering team delivering a well-documented component that fits into a larger system.

The Decision Framework: Choosing the Right Tool for the Task

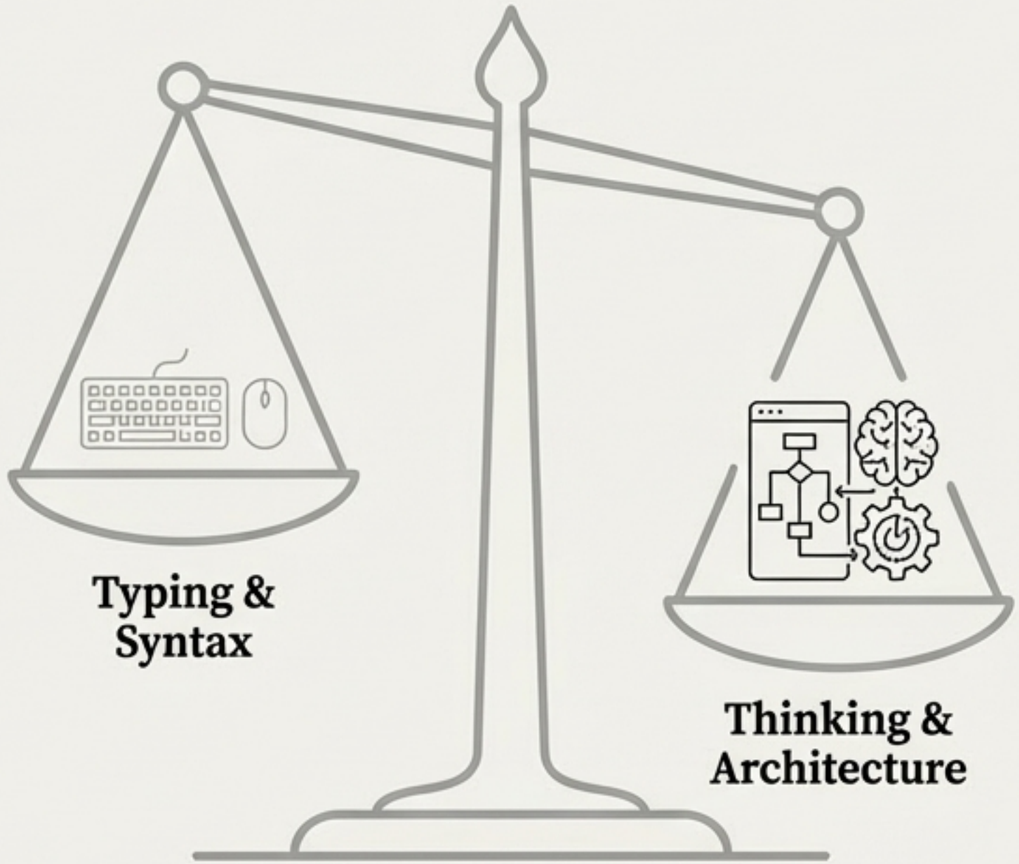


The Emerging Paradigm: Your Role is Evolving

The value of a developer is shifting.

FROM

Writing code line-by-line, mastering syntax, and manual debugging.



TO

Defining problems with precision, reviewing AI-proposed solutions solutions critically, and making high-level architectural decisions.

AI-native IDEs don’t replace developers; they augment them. Your leverage moves from your typing speed to the quality of your thinking and specification. The most effective developers will be the best collaborators with AI.

Your Next Steps to Mastery

1 Pick Your Arena

Based on the decision framework, choose one IDE that best fits a typical project you work on. Don't try to learn all three at once.

2 Start a Pilot Project

Spend one week building a small personal project with your chosen IDE. This could be a simple script, a utility, or refactoring an existing tool.

3 Master the Workflow

Focus on mastering its unique collaboration pattern. Resist the urge to use it like your old editor. If you chose Cursor, embrace the spec-first chat. If you chose Zed, live in the inline assistant. If Antigravity, trust the artifact process.

Resources for Your Journey

- Official Documentation: zed.dev/docs
- Community & Discussion: zed.dev/community

- Official Documentation: cursor.com/docs
- Cursor vs. VS Code: cursor.com/compare

- Official Documentation: antigravity.google.com/docs
- DeepMind Blog (for updates): deepmind.google.com/blog

> The best tool is the one that best amplifies your thinking. Experiment, adapt, and find your ideal AI teammate.