KiloCode – Full System Documentation

This document provides a consolidated overview of how KiloCode works, including architecture, workflows, modes, error handling, configuration, and best practices.

1. System Overview

KiloCode is an autonomous AI coding agent designed to operate inside development environments (such as VS Code). It uses a task-oriented agent architecture combined with modes, tools, memory, and rules to safely and efficiently perform software development tasks.

Core components:

1. LLM Provider Layer – Connects to OpenRouter, OpenAI-compatible APIs, or local models.
2. Agent Modes – Predefined behavior profiles (Code, Debug, Architect, Ask).
3. Tooling Layer – File system access, terminal execution, browser, memory.
4. Rules Engine – Enforces constraints and project standards.
5. Memory Bank – Persistent long-term context.

2. Agent Modes

1. Code Mode – Writes, edits, and refactors code.
2. Debug Mode – Diagnoses errors, stack traces, and runtime issues.
3. Architect Mode – Designs systems and proposes technical plans.
4. Ask Mode – Answers conceptual or technical questions.

3. Workflows

Workflows are markdown-based scripts stored in .kilocode/workflows/. They allow deterministic step-by-step execution of complex tasks such as code generation, audits, or migrations.

4. Skills

Skills are reusable instruction blocks stored in .kilocode/skills/. They extend the agent’s abilities and can be global or project-specific.

5. CLI Usage

1. kilocode run – Executes the agent.
2. kilocode doctor – Diagnostics and environment checks.
3. kilocode update – Updates the agent.

CLI Exit Codes:

1. 0 – Success
2. 1 – General error

3. 124 – Timeout

6. Error Handling

KiloCode handles errors through layered validation, tool fallbacks, and retry logic. Most errors fall into configuration, provider, or tool execution categories.

1. Provider errors – Invalid API keys, rate limits, or model incompatibility.
2. Tool errors – Terminal failures, permission issues, or missing files.
3. Context errors – Overloaded context window or malformed workflows.

7. Rules & Compliance

Rules define strict behavioral constraints and are enforced before output generation. They can control code style, security, compliance, and language usage.

8. Memory Bank

The Memory Bank stores long-term project context and decisions. This improves continuity and reduces repetition across sessions.

9. MCP (Model Context Protocol)

MCP allows KiloCode to interact with external services and internal tools using a standardized interface.

10. Best Practices

1. Use Architect mode before large changes.
2. Define rules early for consistency.
3. Use workflows for repeatable tasks.
4. Monitor CLI exit codes during automation.