The 7-step delivery framework

Integrating architectural thinking with high-velocity automation

codilime

1. Requirements discovery

The engineering process begins by identifying core business logic, edge cases, and non-functional requirements to establish a live "source of truth." Rather than creating static documents that gather dust, we focus on identifying hidden assumptions and defining expectations around scalability, performance, security, and reliability from the outset.

These requirements are structured into narrow, optimized contexts. By doing so, we prevent context overloading and ensure that any subsequent automated workflows remain precise and free of logic hallucinations.

These PRD documents remain live and are updated whenever requirements change, ensuring system knowledge stays consistent and current throughout the project lifecycle.

To accelerate this phase, we use predefined requirement patterns for common features, giving the team a robust starting point and significantly speeding up documentation creation.

2. Architecture setup

Foundational technical decisions, including the stack, integration patterns, and data flow, are captured in a living "Project Memory" document. This serves as the single source of truth that maintains architectural consistency across the entire development team.

At this stage, we establish and document the frameworks and major architectural constraints that shape the system. To enhance this source of truth, we utilize specialized automation to translate technical specifications into UML diagrams. This provides the engineering team with synchronized written guidance and visual models that evolve alongside the project.

The result is a system that is easier to understand, review, and evolve, providing precise architectural views that stay aligned as the codebase grows.

3. Data modeling

Data modeling is a deliberate validation phase where critical domain models are designed manually to secure the persistence layer and business logic. We treat this as one of the most critical steps in the entire process; it is a validation phase rather than a byproduct of implementation.

Before building new features or starting a system from scratch, we perform a thorough review of the domain models. These models are essential because they shape the persistence layer and influence the core business logic.
This stage functions as a primary quality gate: if the domain models are architecturally sound, it confirms the project context is ready for high-speed development.

In this sense, data models serve as an early indicator of how well the project context has been prepared.

4. Backend development

Business logic is kept on the backend to ensure reliable validation through a fast, automated feedback loop. We enforce high modularity to provide development workflows with narrow, component-level context, which significantly increases implementation precision and code quality.

Both the codebase and the "Project Memory" are structured around these independent components. Automated testing is a critical part of this stage; we develop unit tests in parallel to ensure that changes are validated instantly without the need to scan the entire system.

This modular approach improves precision and makes ongoing development more efficient, as developers can quickly run relevant tests and validate impact immediately upon any component change.

5. Frontend implementation

In frontend development, we focus on building the user flow and presentation layer while maintaining a strict separation of concerns by keeping business logic on the backend. This makes the frontend simpler, more predictable, and easier to evolve.

Development is driven by a structured design system and component-based architecture. We have replaced traditional, static design handoffs with clickable prototypes for instant user validation. To ensure consistency and speed, we rely on reusable components and dedicated guidance on how to implement UI in a maintainable way.

Finally, end-to-end testing with Playwright ensures that interfaces remain production-grade and regression-free, delivering a polished user experience rather than a "generated" feel.

6. Quality assurance

Quality assurance in our process goes beyond standard testing. We utilize a full testing pyramid, encompassing unit, integration, and E2E tests, to provide a rapid feedback loop for all implementations. This automated validation extends to non-functional requirements like performance and security, ensuring production-readiness from the very first commit.

Quality is monitored through real-time dashboards tracking trends in cyclomatic complexity, test coverage, and security vulnerabilities. To prevent "mirroring" errors, implementation and verification are often decoupled, using separate automated agents or manual expert reviews to ensure independent validation.

Finally, environment provisioning allows for instant testing in staging setups that precisely mirror real-world production conditions.

7. Infrastructure & CI/CD

In practice, the infrastructure stage starts before the first commit, not at the end of the cycle. Deployment and release pipelines are established upfront, serving as the system's "source of truth" for quality and security from day one.

Alongside each release, we generate detailed quality trend reports based on metrics such as test coverage, cyclomatic complexity, code duplication, and dependency health. By using the same high-quality bars for both automated and manual code, we protect the main branch and keep the system maintainable.

This gives the team continuous visibility into how the system evolves and ensures that speed of delivery never comes at the cost of long-term technical debt.