Persistent
Re(AI)magining™ the World

Agentic Evolution

Software Development in the Era of Autonomy

Optimizing the SDLC Lifecycle with Gemini

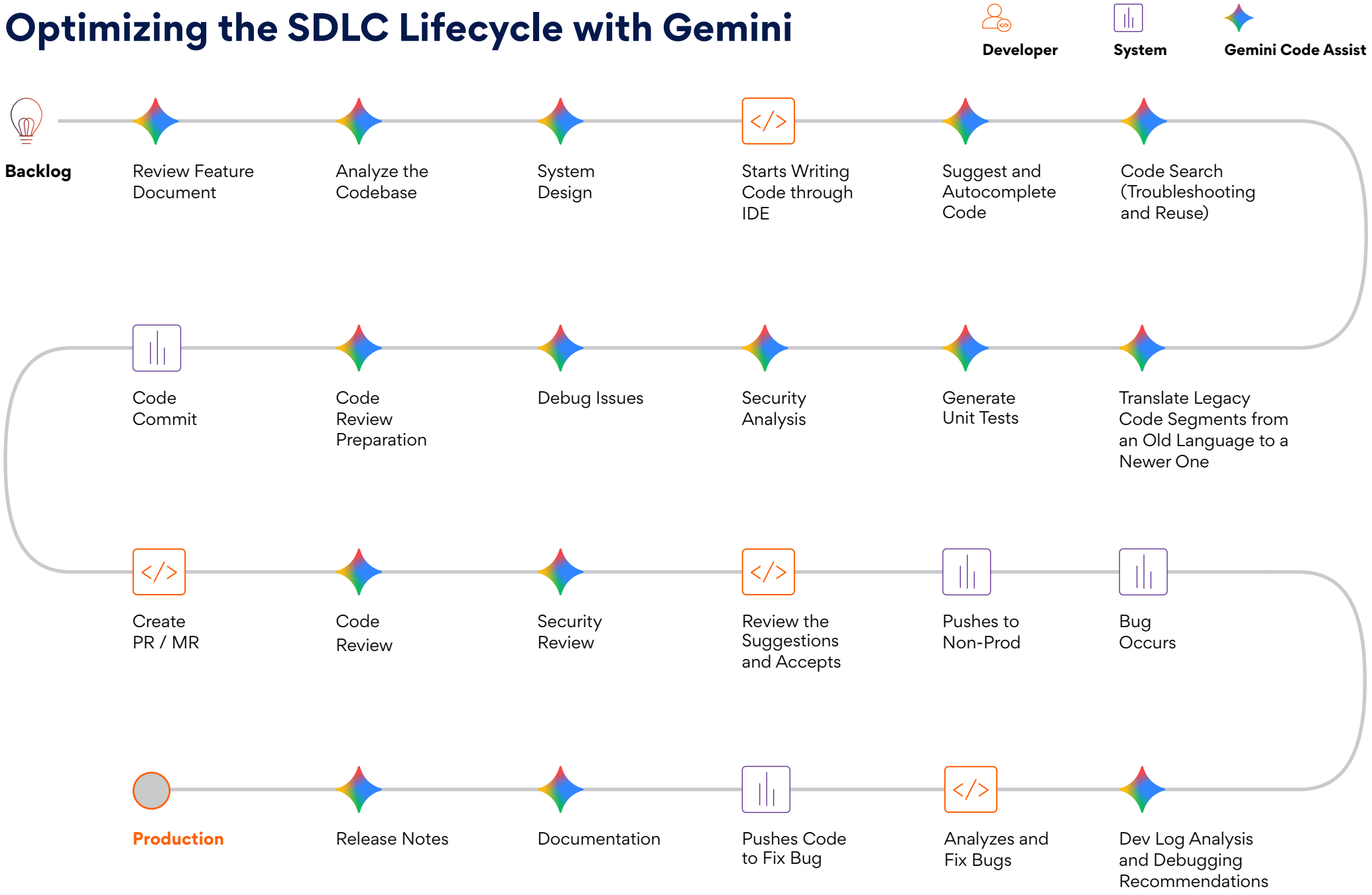


Table of Contents

Chapter 1 | Legacy Drag: Unchanged Core and Cost of SDLC Inertia

Despite nearly five decades of innovation in computing, the foundational structure of the Software Development Lifecycle (SDLC) has shown remarkable resilience to change. Be it the rigid Waterfall model, the iterative cycles of Agile, or the automated pipelines of DevOps, the necessary stages — planning, design, implementation, testing, deployment and maintenance — have remained static.

Even in organizations that have embraced DevOps, engineers still spend enormous time on:

- Writing boilerplate code
- Creating technical documentation
- Managing test cases
- Performing manual code reviews
- Managing CI / CD pipelines
- Addressing security vulnerabilities
- Diagnosing production issues

These activities collectively represent the operational friction of software delivery.

While tools have evolved, the core SDLC processes remain human-orchestrated, limiting scalability and slowing innovation. This structural consistency has accumulated a "legacy drag," transforming what should be a dynamic pipeline into a series of costly bottlenecks.

Sequential bottlenecks and handoff friction

Even in highly optimized Agile and DevOps environments, the fundamental need for human-to-human handoffs persists. Design must be "complete enough" before coding begins and code must be "feature-complete" before full-scale testing. These manual transitions introduce scheduling dependencies, communication overhead and inevitable delays, stalling velocity and increasing time-to-market.

Manual tax on productivity

A massive portion of the SDLC remains stubbornly human-centric. This includes the laborious process of gathering requirements, the meticulous creation and syncing of documentation, the repetitive writing of unit and integration tests, and the complex, error-prone setup of production monitoring and alerting systems. This "tax" diverts millions of development hours away from high-value, creative problem-solving.

Exorbitant cognitive load on developers

Modern software development is inherently complex, involving polyglot codebases, distributed architectures (microservices, serverless), and rapidly evolving Infrastructure-as-Code (IaC) standards. Developers spend an increasing proportion of their time not on feature implementation, but on debugging infrastructure code, searching through siloed documentation, resolving integration failures and wrestling with brittle CI / CD pipeline configurations. This burnout-inducing cognitive load is unsustainable.

Security as late-stage gating factor

While the industry has widely adopted the principle of "shift-left security," in practice, security often remains a distinct, non-integrated phase. Late-stage Static Analysis (SAST) or Dynamic Analysis (DAST) findings force costly, disruptive remediation efforts just before release, turning security from a preventative measure into a project blocker.

This structural inertia has imposed a ceiling on true organizational speed and efficiency. The SDLC did not merely need optimization; it required a complete architectural paradigm shift.

Chapter 2 | Agentic AI Era: Architecting the Future of SDLC

The emergence of Generative AI (GenAI), specifically the advancement of Agentic Systems, represents the first true, fundamental transformation of the SDLC model. In 2026, we have moved past the "Copilot" era (assistive) into the "Agent" era (executive).

Agentic AI moves beyond simple code completion or text generation; it introduces interconnected, goal-oriented software entities, known as agents, that possess the ability to:

Reason

Understand complex objectives and constraints

Plan

Breakdown high-level goals into executable sub-tasks

Execute

Interact with tools, APIs and the codebase to perform actions

Reflect and Iterate

Self-critique results and adjust the plan if execution fails

In this new paradigm, the SDLC shifts dramatically from a human-driven, tool-supported process to an AI-orchestrated, human-supervised process. The developer's role evolves from the "hands-on builder" to the "chief architect and system supervisor," setting objectives and validating agent performance.

The driving force of this transformation is end-to-end orchestration and automation, in which specialized agents autonomously manage, coordinate and execute the flow across all traditional SDLC stages.

Traditional Human-Centric SDLC

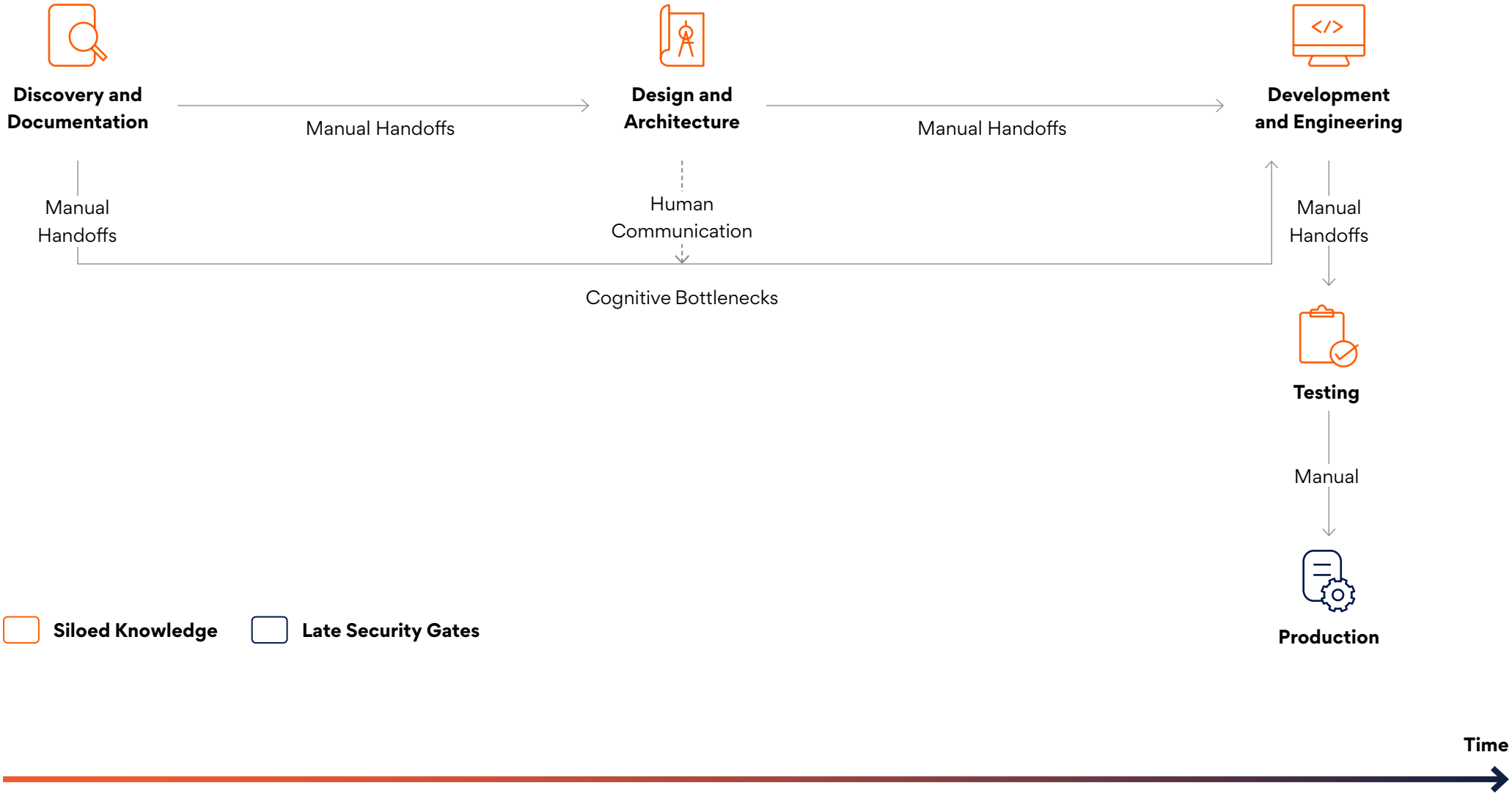
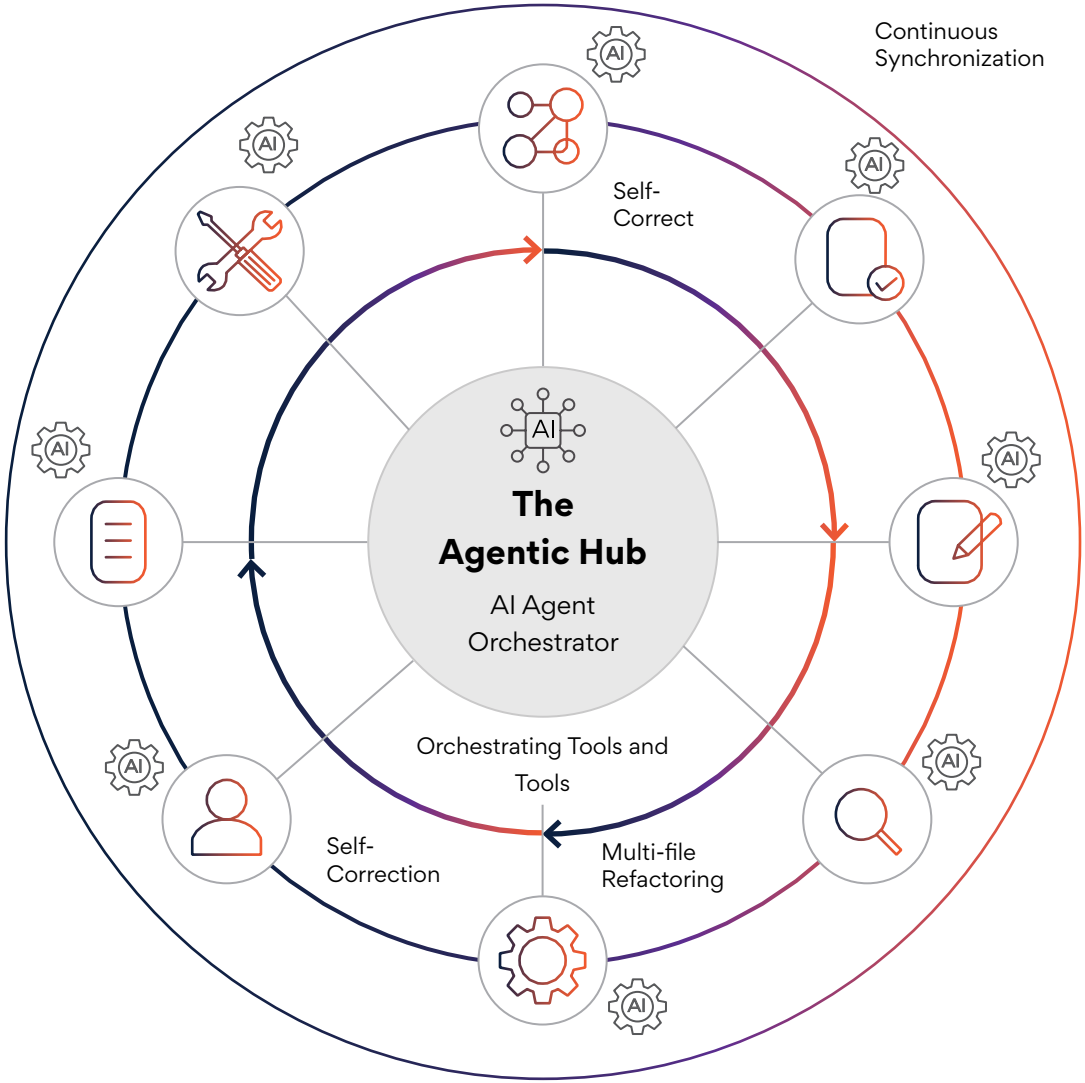


Figure 1A: From Manual SDLC

Agentic AI-Native SDLC

Figure 1B: Autonomous Agentic SDLC — Orchestrated by the Agentic Hub



Chapter 3 | Deep Transformation Across SDLC Stages

The Agentic SDLC embeds autonomous, context-aware agents into every phase, eliminating friction and maximizing velocity.

Design and Discovery: Requirement Agent

This initial stage moves from ambiguity to instant, actionable clarity.

Current State (Manual)	Agentic Transformation (Autonomous)	Key Agent Role: Requirement Agent
Time-consuming stakeholder interviews and manual note-taking	Continuous, context-aware requirement synthesis and validation	**Input analysis and synthesis:** The requirement agent ingests and analyzes unstructured data, such as customer support tickets (e.g., Zendesk), sales call transcripts (e.g., Gong), market research and existing product analytics, to synthesize precise, validated user stories and detailed acceptance criteria in real-time.
Creation of static mockups, often leading to developer misinterpretation	Generation of executable prototypes, dynamic UI code snippets and design system adherence	**Design-agent collaboration:** Converts high-level requirements into functional wireframes and generates corresponding front-end code (e.g., React or Vue components) that strictly adhere to the organization's codified design system and accessibility standards, eliminating the front-end translation bottleneck.
	Automated dependency mapping and conflict detection	**Conflict resolver:** Automatically identifies potential dependencies, system conflicts and technical debt implications associated with new features, flagging them to the Architect Agent before coding begins.

Documentation and Design: Architect Agent

The Architect Agent functions as the central intelligence and knowledge hub, ensuring design integrity and preventing “architecture drift.”

- **System architecture generation:** Based on the validated requirements, the agent dynamically proposes optimal technology stacks, defines secure microservice boundaries, selects appropriate data stores and suggests efficient deployment patterns (e.g., Kubernetes vs. Serverless). It generates complete, runnable Infrastructure-as-Code (IaC) templates (Terraform, CloudFormation, Bicep) ready for deployment.
- **Living documentation:** In a radical departure from traditional documentation, every code change, feature addition, configuration update or deployment adjustment automatically triggers a corresponding documentation update. The agent maintains:
  - Real-time API specifications (OpenAPI / Swagger)
  - Up-to-date system topology diagrams (e.g., C4 model)
  - Functional guides and operational runbooks
- **Design validation and governance:** The agent continuously monitors code committed by the Coding Agent, ensuring it strictly adheres to the proposed architectural design and organizational governance policies, flagging “design drift” immediately and preventing technical debt accumulation.

Development and Engineering: Coding Agent

The implementation phase moves from human-intensive coding to AI-driven code generation, refinement and adaptation.

Native IDE integration and contextual awareness

The Coding Agent is not a simple suggestion tool (like initial LLM assistants); it is a fully capable participant, natively integrated within the IDE environment. It operates with complete contextual awareness of the entire repository, existing APIs, dependencies and style guides.

Goal-oriented feature generation

A developer provides the objective (e.g., “Implement the federated user login flow using SAML and integrate it with the Billing Service API”), and the agent orchestrates the necessary code changes across all affected layers: front-end UI, back-end business logic, database migration and even environment configuration files.

Refinement, optimization and compliance

The agent proactively proposes code refactoring to improve performance, reduce complexity, enhance security and enhance maintainability, ensuring the generated code adheres to organizational style guides, language-specific best practices and API versioning standards.

Testing and Quality Assurance: QA and Security Agent

Testing is transformed from a sequential, time-consuming step into a parallel, continuous and integrated activity.

Testing Domain	Unit and Integration Testing	Performance / Load Testing	Shift-left Security
Agentic Transformation	Automatic, comprehensive test case generation and self-healing test maintenance.	Continuous, realistic load profile generation and analysis.	Proactive, pre-commit vulnerability identification and remediation. **Code Scan:** Uses services like SonarQube, Checkmarx or internal LLM-powered logic to scan new code for common vulnerabilities (XSS, SQLi, injection flaws) and bad practices before the code is merged (e.g., via pre-commit hooks or as the first step in the CI pipeline). **Container Scan:** Integrates with tools like Wiz or Tenable to scan Docker images and registries for known Common Vulnerabilities and Exposures (CVEs) and severe misconfigurations during the build phase within CI / CD.
Key Agent Role: The QA Agent and Security Agent	**QA Agent:** Generates comprehensive unit, integration and end-to-end test suites based directly on the Requirement Agent's acceptance criteria. Crucially, it automatically updates or regenerates failing tests when validated code changes are introduced, eliminating the perennial problem of test maintenance overhead.	**Performance Agent:** Simulates production-like traffic patterns derived from real user data, identifies granular performance bottlenecks, and collaborates with the Architect Agent to suggest and implement scaling solutions (e.g., resource pooling changes, database index optimization, cache layer modifications).	**Security Agent:** Executes continuous scans (Static Analysis Security Testing — SAST, Dynamic Analysis — DAST, and IaC security checks) and deeply integrates with industry-leading services.

Productionizing Workloads: Deployment and Monitoring Agent

This final stage is managed by an agent dedicated to ensuring secure, observable and resilient application deployment.

CI / CD orchestration and release management

The Deployment Agent manages the entire CI / CD lifecycle. It intelligently triggers the build process, verifies that all security gates (code, container and IaC scans) orchestrated by the Security Agent have passed, manages environment variables and secrets, and executes sophisticated deployment strategies (e.g., automated blue / green or canary rollouts).

Infrastructure management

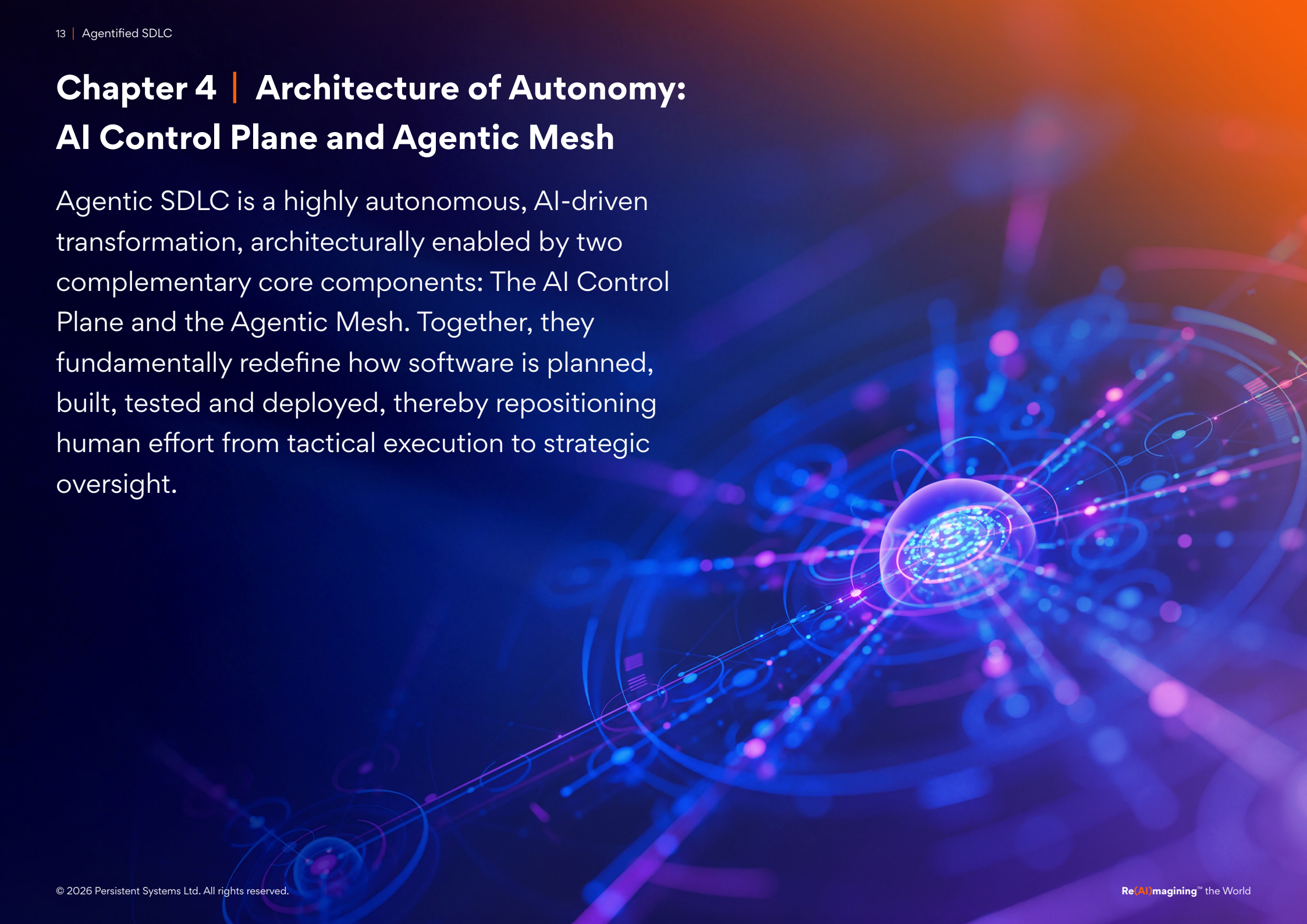
This agent uses IaC generated by the Architect Agent to provision, update and de-provision cloud resources (on AWS, Azure, GCP, etc.), ensuring infrastructure parity and adherence to security best practices (e.g., least-privilege access).

Automated observability setup

The agent automatically instruments the newly deployed application with necessary monitoring, logging and tracing tools (e.g., Prometheus, Grafana, OpenTelemetry, Datadog). Crucially, it defines relevant alerts based on codified Service Level Objectives (SLOs) derived directly from the initial requirements, ensuring proactive operations from day one.

Chapter 4 | Architecture of Autonomy: AI Control Plane and Agentic Mesh

Agentic SDLC is a highly autonomous, AI-driven transformation, architecturally enabled by two complementary core components: The AI Control Plane and the Agentic Mesh. Together, they fundamentally redefine how software is planned, built, tested and deployed, thereby repositioning human effort from tactical execution to strategic oversight.



AI Control Plane: Central Intelligence and Governance Layer

The AI Control Plane functions as the enterprise-level "brain" of the Agentic SDLC. It serves as the single source of truth for objectives, policies, resources and the overall system state. Its mandate is not the execution of development tasks but rather the orchestration, management and governance of the autonomous agents that perform them.

Expanded core functions

Unified policy and guardrails enforcement

This mechanism serves as the critical safety layer. The Control Plane enforces non-negotiable organizational rules, including:

- **Security policies:** Mandating the use of approved libraries, adherence to OWASP guidelines and continuous security scanning protocols.
- **Compliance:** Ensuring code and documentation meet regulatory standards (e.g., GDPR, HIPAA).
- **Ethical and brand guardrails:** Preventing agents from generating biased, harmful or off-brand content in documentation, code comments or user interfaces.

Strategic goal setting and decomposition

This function extends beyond mere task breakdown. The Control Plane ingests high-level business goals (e.g., "Increase conversion rate by 10% in Q3") and employs sophisticated reasoning to generate a hierarchical, executable development plan. It translates strategic objectives into concrete, measurable user stories, tasks and sub-tasks, complete with success criteria and dependency mapping.

Dynamic resource and context management

The Control Plane optimizes the utilization of expensive computational resources. It dynamically allocates specific LLMs (GPT-4, Claude and domain-specific models) and hardware (GPUs) to agents based on the task's complexity and security requirements. Furthermore, it manages the context window for agents, ensuring access to the relevant code history, architectural documents and external APIs without exceeding their processing capacity.

Comprehensive monitoring, observability and auditing

The Control Plane provides a transparent, end-to-end view of the Agentic SDLC. It meticulously tracks agent performance metrics (e.g., lines of code generated per hour, bug introduction rate, task completion time) and generates detailed audit logs, which are crucial for compliance and post-mortem analysis.

Agentic Mesh: Distributed Execution Network

The Agentic Mesh is the decentralized, dynamic network of specialized, autonomous AI entities. Each agent functions as a microservice of intelligence, equipped with domain expertise, tools and the capability to act independently to achieve goals assigned by the Control Plane. It constitutes the “hands and feet” of the development process, operating with high parallelism.

Elaborated core characteristics

Deep specialization and persona

Agents are architected around specific software development personas and technical domains, such as:

- **System Architect Agent:** Focused on generating and validating high-level designs and IaC.
- **Refactoring Agent:** Constantly monitors the codebase for technical debt and proactively proposes or executes refactoring tasks.
- **Integration Agent:** Specializes in generating glue code and ensuring API compatibility between microservices.
- **Documentation Agent:** Maintains live, up-to-date documentation synchronized with every code change.

Advanced collaboration and artifact management

Agents communicate through a structured, standardized protocol, frequently exchanging serialized data artifacts (e.g., JSON specification files, validated code snippets, test reports). This “hand-off” process is self-correcting: should the Testing Agent detect a bug, it automatically transmits a detailed failure report back to the relevant Code Generation Agent for immediate, human-free remediation.

Tool autonomy and adaptability

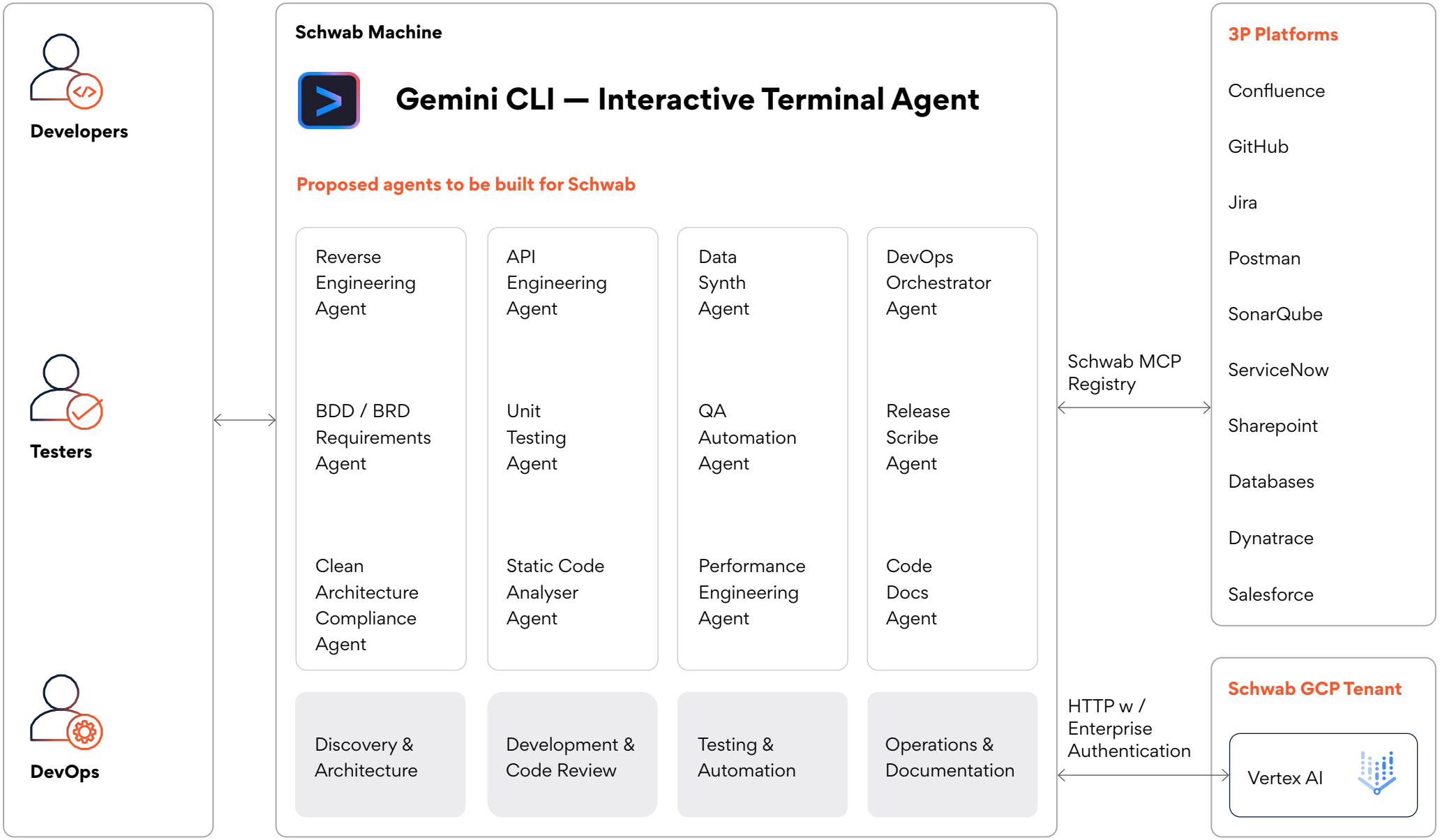
Agents are empowered to utilize a rich set of external tools. This involves not simple API calling, but tool reasoning — the capacity to determine which tool to use (e.g., Git, Docker, Kubernetes, specific testing frameworks or proprietary internal systems) and how to use it effectively to resolve a problem, reflecting a dynamic, goal-oriented approach

SDLC Transformation: From Assembly Line to Ecosystem

The Control Plane and Mesh eliminate the linear boundaries of traditional SDLC, forming a continuous and self-optimizing ecosystem.

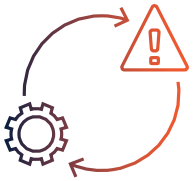
SDLC Phase	Agentic SDLC (AI / Agent-Centric) Transformation and Impact
Planning and requirements	The Control Plane initiates development by converting ambiguous goals into formal, executable specifications. The Requirements Agent utilizes knowledge of past projects and user feedback to ensure specifications are complete, consistent and traceable, virtually eliminating ambiguity before coding commences.
Design and architecture	Design Agents generate multiple, optimized architectural proposals based on performance, cost and security constraints provided by the Control Plane. Human architects validate the final choice, dramatically curtailing time expenditures on initial modeling and documentation.
Implementation (coding)	Code Generation Agents and Review Agents operate in parallel, committing production-ready, highly optimized code. This phase runs continuously, driven by the Mesh. The human role shifts entirely to code validation (ensuring the code meets the intended purpose) rather than the act of writing the code itself.
Testing and quality assurance	The Test Agent not only generates test cases but executes self-healing testing: Dynamically updating tests when code changes and autonomously debugging its own generated failures. The Security Agent ensures a “secure by default” posture through continuous, granular enforcement within the development pipeline.
Deployment and operations	The Deployment Agent manages a fully autonomous CI / CD pipeline. The Control Plane’s policies dictate release cadence and risk tolerance, allowing the agent to automatically manage canary deployments, gradual rollouts and infrastructure scaling with minimal human oversight.

Transforming SDLC with Google



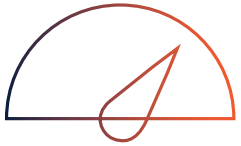
AI Control Plane and Agentic Mesh as Core Pillars

These two components are not merely supplementary features; they represent the enabling structure for truly autonomous software creation.



Orchestration, coherence and risk mitigation

The Control Plane preempts the “tyranny of autonomy.” By maintaining a single, global view of the development goal and enforcing strict guardrails, it ensures that while agents are autonomous, their combined efforts remain coherent, preventing agents from operating at cross-purposes, introducing technical debt or violating enterprise policies.



Hyper-scalability and acceleration

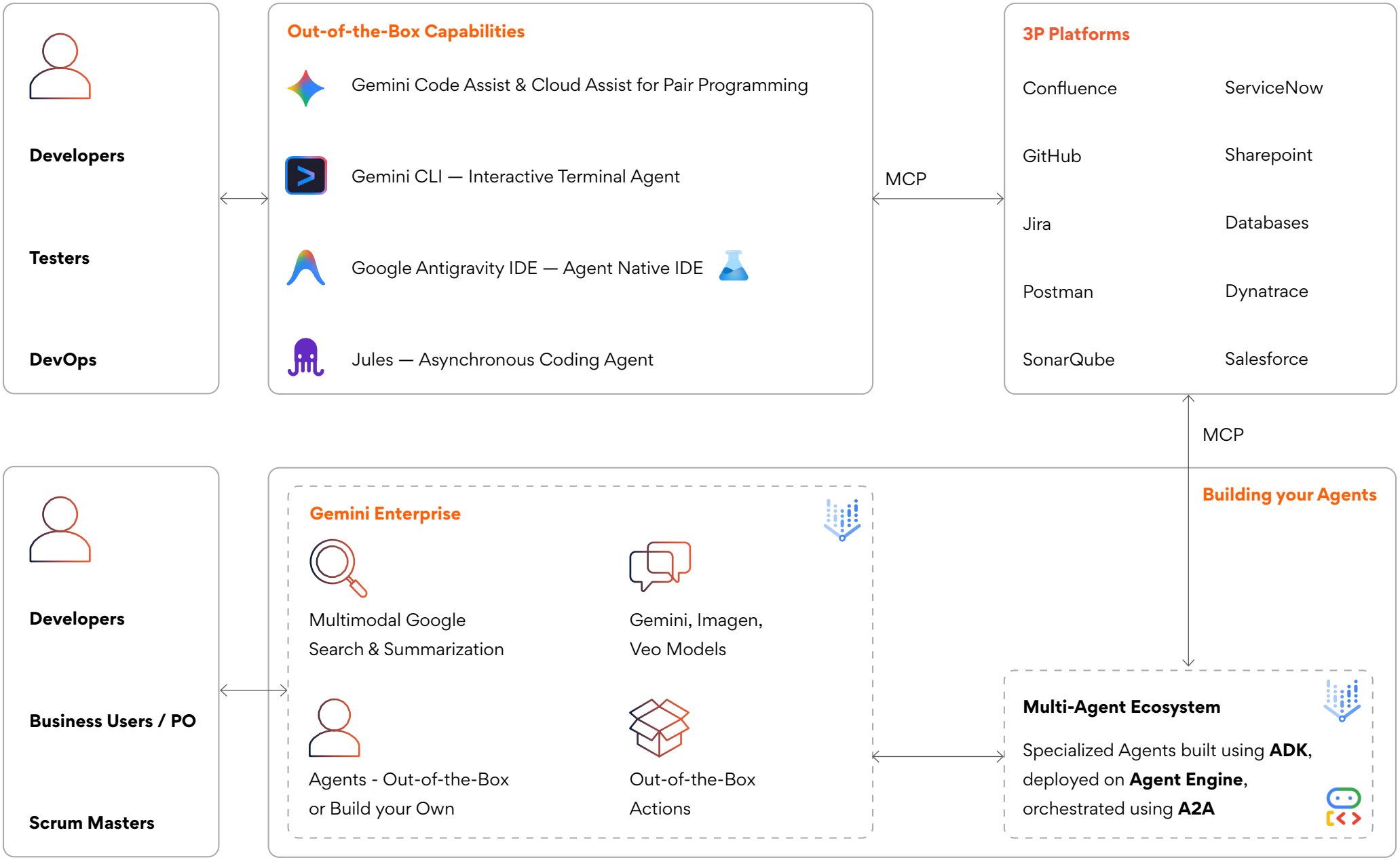
The Agentic Mesh enables massive concurrency, running hundreds of tasks simultaneously. The Control Plane optimizes execution, leading to unprecedented development speed. Features that once required weeks can be planned, built and tested in days, resulting in hyper-productivity.

Human empowerment and strategic focus

This architecture fundamentally alters the role of human developers. They are liberated from mundane, repetitive coding and QA tasks. Their expertise is redirected toward defining the Control Plane’s strategic goals and policies, validating AI-generated outcomes and architecting the next generation of autonomous agents.

In summary, the AI Control Plane provides intelligent governance and direction, while the Agentic Mesh provides specialized, parallel execution power. This synergy unlocks the full potential of the Agentic SDLC, transforming software development into an intelligent, self-driving ecosystem.

Figure 2: The Agentic SDLC



Chapter 5 | Native Integrations: Connected Fabric

One of the most profound transformations introduced by the Agentic SDLC is the deep integration of AI agents directly within the developer ecosystem.

Traditional software delivery environments consist of loosely coupled tools connected through APIs, webhooks and manual triggers. Developers switch between these tools throughout the day, leading to significant context switching and productivity loss.

The Agentic SDLC eliminates this fragmentation by embedding intelligent agents directly into the core environments where software is designed, developed, built, tested and deployed.

In this model, AI agents operate not as external assistants but as native participants within the engineering workflow.

These agents observe development activity, understand system context and autonomously orchestrate tasks across the lifecycle.

The three primary integration layers includes

- Developer IDE environments
- Continuous Integration / Continuous Deployment (CI / CD) pipelines
- Security and compliance scanning frameworks

Together, these layers create a fully integrated autonomous engineering fabric. Agentic SDLC relies heavily on deep native integration with developer environments.

IDE Integration

The Integrated Development Environment (IDE) is the primary workspace for software engineers. However, traditional IDEs have functioned mainly as code editors and debugging tools, leaving most lifecycle activities to external platforms.

The Agentic SDLC transforms the IDE into an AI-native engineering cockpit, where agents actively collaborate with developers during the coding process.

AI agents operate directly within development environments such as:

- Visual Studio Code
- JetBrains IDEs
- Cloud Development Environments

Within these environments, AI agents perform several autonomous capabilities.

Context-Aware Code Generation

Traditional coding assistants generate code snippets from short prompts. Agentic development systems go significantly further by understanding the entire repository context, architecture patterns and existing service interactions.

Capabilities includes

- Generating entire microservices
- Implementing API layers
- Creating infrastructure configuration
- Writing data access layers
- Generating event-driven workflows

Example workflow

- Developer describes features in natural language
- Agent analyzes the repository architecture
- Agent generates the service implementation
- Agent generates unit tests automatically
- Agent suggests integration tests

The developer becomes the supervisor of generated logic rather than the author of every line of code.

Intelligent Code Refactoring

Large enterprise codebases often accumulate technical debt over time. Agentic IDE integration enables agents to continuously analyze code quality and recommend structural improvements.

Capabilities includes

- Removing duplicated code
- Optimizing inefficient algorithms
- Modernizing legacy frameworks
- Migrating outdated libraries
- Converting monolith components into microservices

Integration with static analysis tools such as, SonarQube allows the agent to automatically resolve code smells and maintain high code quality standards.

Example

If a codebase violates maintainability rules, the agent can:

- Refactor the module
- Update dependent components
- Run automated tests
- Submit a pull request for approval

Autonomous Debugging and Root Cause Analysis

Debugging traditionally consumes a significant portion of engineering time. Agentic IDE environments can analyze logs, stack traces and runtime behavior to automatically diagnose issues.

Capabilities includes

- Stack trace interpretation
- Memory leak detection
- Concurrency issue detection
- Dependency conflicts
- Performance bottleneck analysis

Agents can also reproduce issues locally by simulating runtime conditions and generating debugging reports.

Automated Test Generation

Testing has historically required manual test case creation, which limits coverage. Agentic IDE integration automatically generates multiple layers of testing, including:

- Unit tests
- Integration tests
- API contract tests
- Performance tests
- Edge case validation

Agents analyze function logic and automatically create test scenarios to maximize code coverage. This ensures testing is embedded directly into the development process rather than treated as a downstream activity.

CI / CD Integration

CI / CD pipelines constitute indispensable components of contemporary software delivery methodologies. Despite their inherent criticality, CI / CD pipelines often require substantial manual configuration and ongoing maintenance.

The Agentic SDLC introduces autonomous pipeline orchestration, in which intelligent agents dynamically configure, optimize and maintain the necessary build and deployment infrastructure.

Widely utilized CI / CD platforms includes:

- GitHub Actions
- GitLab CI / CD
- Jenkins
- Argo CD

AI agents seamlessly integrate into these pipelines to automate various stages, encompassing, but not restricted to, the following functions.

Intelligent Pipeline Creation

Agents possess the inherent capability to automatically generate CI / CD workflows from repository configurations and subsequently optimize the resulting build processes.

Specifically

Upon the initiation of a new service, the agent can autonomously generate the mandatory elements:

- Build pipelines
- Container construction stages
- Security scanning stages
- Automated validation workflows
- Deployment workflows

This capability effectively obviates the necessity for manual pipeline engineering intervention.

Build Optimization

Large-scale enterprise pipelines routinely encounter inefficiencies within their build processes. AI agents can enhance pipeline performance through methodical application of:

- Parallelization of discrete build steps
- Advanced dependency caching mechanisms
- Identification and subsequent elimination of redundant stages
- Reduction of container image dimensionality

These optimizations yield considerable reductions in build execution time and associated infrastructure expenditure.

Automated Deployment Orchestration

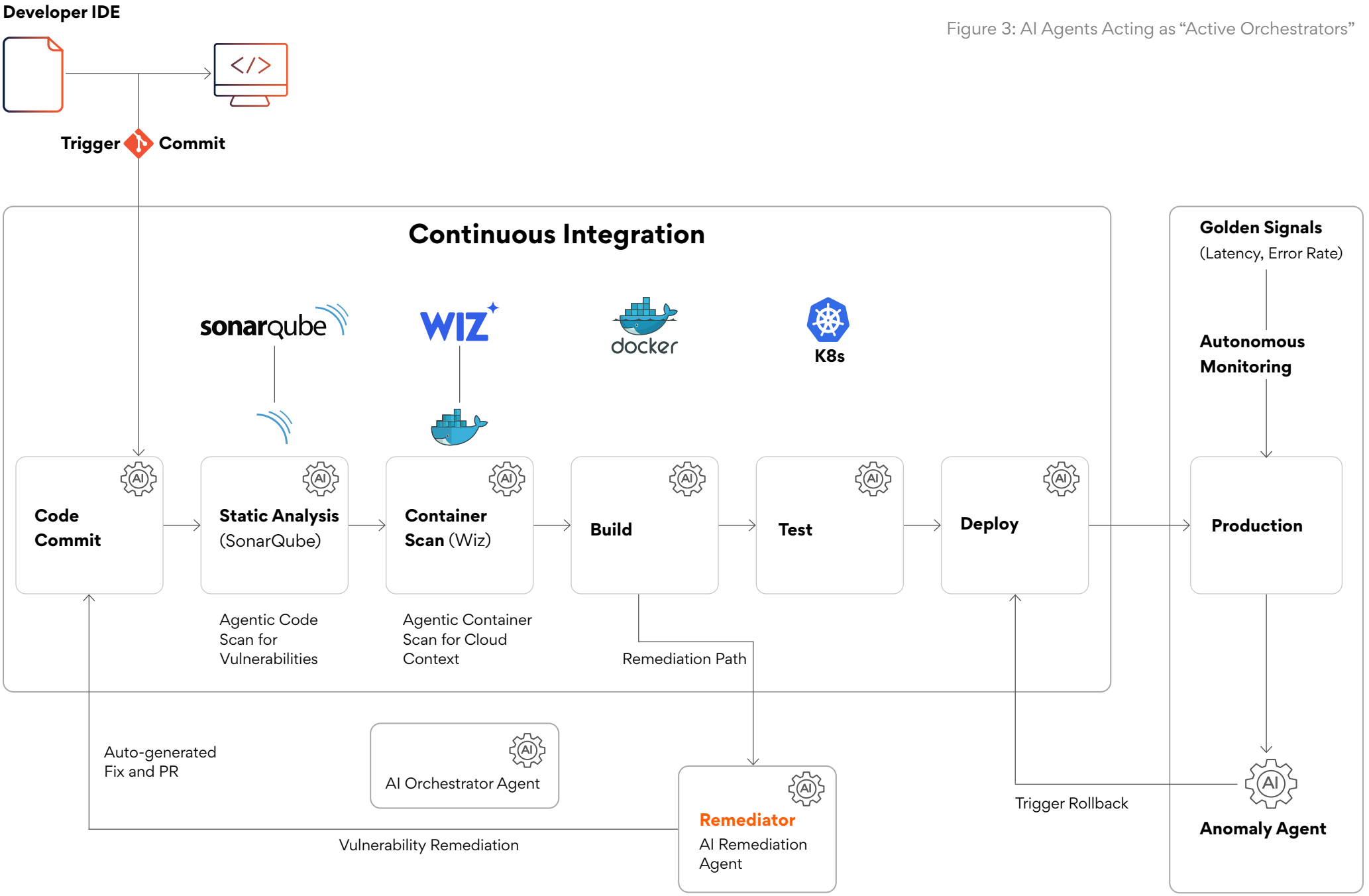
Deployment pipelines mandate meticulous configuration, particularly across segregated staging and production environments.

Agentic pipelines facilitate the dynamic orchestration of sophisticated deployment strategies, such as:

- Canary deployments
- Blue-green deployments
- Feature flag rollouts
- Automatic rollback policies

Agents continuously monitor system performance metrics after deployment and autonomously initiate reversions should operational anomalies be detected.

Figure 3: AI Agents Acting as “Active Orchestrators”



Chapter 6 | Automated Vulnerability Remediation: Shift from Identification to Self-Healing

The true transformative power of the Agentic SDLC is the synchronized flow between specialized agents and the revolutionary capability for automated remediation. The Security Agent is no longer just an alert system; it is an active participant in remediation.

Legacy security was reactive. Agentic security is proactive and remediative.

Code and Container Scanning

Agents integrate natively with services such as SonarQube (for static code analysis / SAST) and Wiz (for cloud / container security).

Scan

As soon as an agent writes code, it triggers a SonarQube scan to check for SQL injection or hardcoded secrets.

Context

Using Wiz, agents examine the cloud context, where they don't just see a vulnerable library; they also check whether that library could be exposed to the internet in the current Kubernetes cluster.

Final Frontier: End-to-End Remediation

The "Agentic Difference" is that the AI does not just flag vulnerabilities; it remediates them.

1. **Identify and patch:** The Security Agent, with Wiz, continuously monitors the CI / CD environment and identifies a critical CVE in a container-based image, detects a security flaw (e.g., an outdated, vulnerable library, a specific insecure coding pattern or a critical IaC misconfiguration), and automatically patches it. Using its deep contextual knowledge and access to security best practices, it autonomously generates the necessary code changes or configuration updates.
2. **Self-correction closed loop:**
    - The Security Agent automatically packages this generated fix (e.g., a patch file, a dependency version bump in package.json or pom.xml or an updated IaC resource definition).
    - This fix is prioritized and submitted back into the development stream as an automated pull request (PR) or patch.
    - The Coding Agent validates the generated fix for code style and function.
    - The QA Agent runs regression tests against the patched code to ensure the fix did not introduce new bugs.
    - The Security Agent re-scans the patched code to verify the vulnerability is fully closed and the pipeline can proceed.

This seamless, closed-loop system dramatically reduces human intervention, narrows the window of vulnerability and moves the SDLC into a truly proactive, self-healing state, fundamentally accelerating software delivery while elevating both security and quality standards.

We have moved from alerting (human-in-the-loop) to auto-patching (human-on-the-loop).

Technical Architecture: SonarQube vs. StreasWiic — Agentic SDLC

Code Commit (IDE)

Static Analysis (SonarQube)
Agentic Code Scan

Validate Fix (Tests & Re-scan)
Integrated Testing

Vulnerability Detected (SAST)
SAST Finding

Generate Fix & Pull Request
Integrated Testing

Container Build & Scan (Wiz & Docker)
Agentic Container Scan

Automated Analysis and Planning
Context Gathering from Wiz
Prioritize Based on Reachability

Vulnerability Detected (Image)
Image Finding

Continuous Remediation Cycle

Figure 4: Technical Architecture of the 'Continuous Remediation Cycle'

Chapter 7 | Road Ahead: Autonomous Software Factories

The shift from traditional SDLC models to Autonomous Software Factories represents one of the most profound transformations in the history of software engineering. Just as the Industrial Revolution automated manufacturing, Agentic AI will fundamentally redefine how software is designed, built, secured and operated.

The Autonomous Software Factory is an environment where software systems are largely self-designing, self-building and self-healing. The focus shifts from the mechanics of coding to the logic of system design and the oversight of agentic workflows. The software engineer is no longer a "writer of lines" but an "orchestrator of intent."

The "Legacy Drag" is being replaced by a frictionless, self-healing and secure-by-default ecosystem where the SDLC is not a process you follow, but an engine that runs itself.

Engineering teams will focus less on writing code and more on:

Supervising AI-driven engineering systems

Designing architecture strategies

Defining governance frameworks

Optimizing engineering platforms

Organizations that successfully transition through these five stages will gain significant advantages in:

Engineering productivity

Operational resilience

Security posture

Innovation velocity

Re(AI)magining™ the World

To streamline your transition to an autonomous software factory and to onboard Agentic SDLC, get in touch with Persistent at grow_gcp@persistent.com.

About Persistent

Persistent Systems (BSE: 533179 and NSE: PERSISTENT) is a global services and solutions company delivering AI-led, platform driven Digital Engineering and Enterprise Modernization to businesses across industries. With over 26,500 employees located in 18 countries, the Company is committed to innovation and client success. Persistent offers a comprehensive suite of services, including software engineering, product development, data and analytics, CX transformation, cloud computing and intelligent automation. The Company is part of the MSCI India Index and is included in key indices of the National Stock Exchange of India, including the Nifty Midcap 50, Nifty IT and Nifty MidCap Liquid 15, as well as several on the BSE such as the S&P BSE 100 and S&P BSE SENSEX Next 50. Persistent is also a constituent of the Dow Jones Sustainability World Index. The Company has achieved carbon neutrality, reinforcing its commitment to sustainability and responsible business practices. Persistent has also been named one of America's Greatest Workplaces for Inclusion & Diversity 2025 by Newsweek and Plant A Insights Group. As a participant of the United Nations Global Compact, the Company is committed to aligning strategies and operations with universal principles on human rights, labor, environment and anti-corruption, as well as take actions that advance societal goals. With 468% growth in brand value since 2020, Persistent is the fastest-growing IT services brand in 'Brand Finance India 100' 2025 Report.

USA
Persistent Systems, Inc., 2055 Laurelwood Road, Suite 210, Santa Clara,
CA 95054, Tel: +1 (408) 216 7010, Fax: +1 (408) 451 9177,
Email: info@persistent.com

India
Persistent Systems Limited, Bhageerath, 402, Senapati Bapat Road,
Pune 411016, Tel: +91 (20) 6703 0000,
Fax: +91 (20) 6703 0008

Persistent
Re(AI)magining™ the World

www.persistent.com