AI-Assisted Monolith to Microservices Transformation

Accelerating Legacy Modernization with the Cadmus Logic.AI^SM ReGenX Platform

Executive Summary

Breaking apart a monolithic application into microservices has always been one of the hardest problems in software architecture. Teams spend months analyzing code, debating boundaries, and worrying about what they might miss. This whitepaper explains the core challenges of monolith to microservices transformation and introduces Cadmus Logic.AI^SM ReGenX, an AI-assisted modernization platform that transforms monolith code bases regardless of size or complexity. It walks through the five-phase methodology that underpins the platform. It describes how ReGenX analyzes code, discovers domains, plans migrations, defines services, and produces implementable microservices in a language and framework agnostic manner.

CADMUS

It also shows how artificial intelligence (AI) is integrated into key decision points and how human expertise remains central to the process. Rather than replacing architects, ReGenX augments their work. It automates deep analysis of existing systems, proposes domain boundaries, designs migration plans and generates detailed service definitions and code. At every step, humans stay in control and use the platform's output as a starting point to make informed decisions.

“ At every step, humans stay in control and use the platform's output as a starting point to make informed decisions.

The Challenge: Why Microservices Transformations Fail

Most organizations that attempt a microservices migration encounter similar obstacles. Code analysis takes weeks. Domain boundaries remain vague. Migration plans fail to capture real dependencies and risks. Service definitions often leave out critical implementation details until late in the project when changes are expensive.

The core problem is not a lack of expertise. Experienced architects still struggle with the sheer volume of information in large codebases and extensive documentation. They must track thousands of classes, follow long dependency chains, and understand how data flows through the system. Doing this work by hand is time consuming and error prone.

ReGenX addresses this challenge by embedding an AI-assisted, five-phase methodology into a single modernization platform. It helps teams turn a complex, opaque monolith into a set of well-defined microservices backed by a clear, data-driven migration plan.

Introducing Cadmus Logic.AI^{SM} ReGenX

ReGenX is a modernization platform that implements a complete five phase approach for transforming monolithic applications into microservices. It connects deep static analysis, domain driven discovery, migration planning, service definition, and code generation into one coherent workflow.

The platform is built around a simple idea. A transformation of this scale should be guided by data and patterns, not just by intuition. Each phase produces structured, machine-readable output that feeds the next phase and that can also be inspected and refined by humans.

Within ReGenX, each phase is implemented as an engine that consumes and produces JSON-based artifacts. These artifacts capture the monolith's structure, proposed domains, migration waves, service specifications, and generated code mappings. This design allows teams to run phases iteratively, adjust decisions earlier, and keep a clear trace of how the architecture evolves over time.

“ A transformation of this scale should be guided by data and patterns, not just by intuition. Each phase produces structured, machine-readable output that feeds the next phase and that can also be inspected and refined by humans.

The Five Phase Methodology Inside the Platform

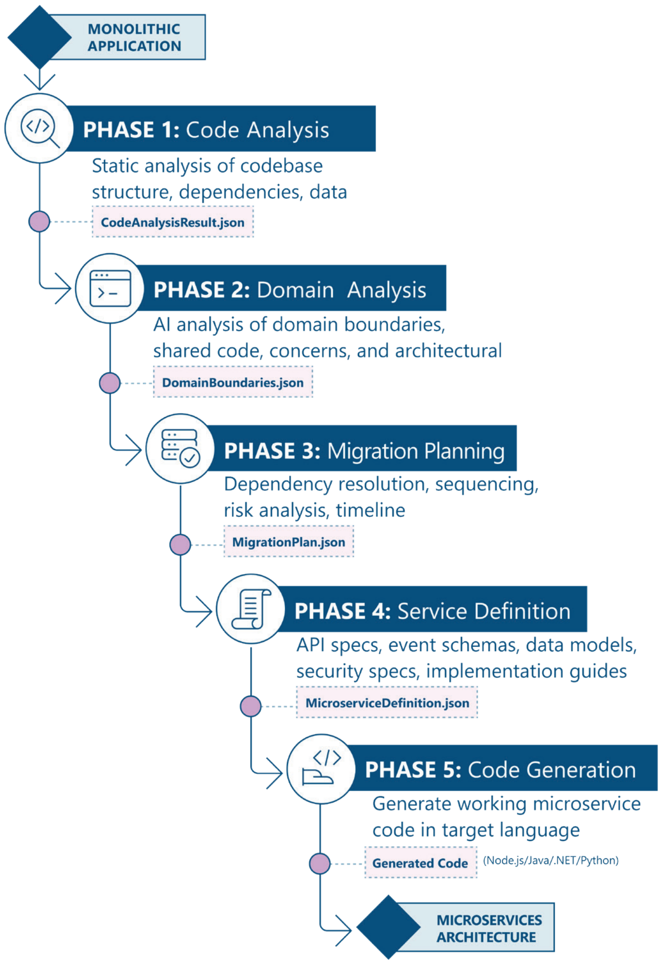

MONOLITHIC APPLICATION
PHASE 1: Code Analysis
Static analysis of codebase structure, dependencies, data
CodeAnalysisResult.json
PHASE 2: Domain Analysis
AI analysis of domain boundaries, shared code, concerns, and architectural
DomainBoundaries.json
PHASE 3: Migration Planning
Dependency resolution, sequencing, risk analysis, timeline
MigrationPlan.json
PHASE 4: Service Definition
API specs, event schemas, data models, security specs, implementation guides
MicroserviceDefinition.json
PHASE 5: Code Generation
Generate working microservice code in target language
Generated Code
(Node.js/Java/.NET/Python)
MICROSERVICES ARCHITECTURE

Figure 1. Cadmus Logic.AI^{SM} ReGenX Five-Phase Methodology

The core of ReGenX is a five-phase methodology that breaks the transformation into manageable steps. The phases are not rigid. Teams can move back and forth between them as they learn more about the system. What follows in Figure 1 is a look at each phase and how the platform supports it.

Phase 1: Code Analysis

The journey begins with a clear understanding of the system as it exists today. In Phase 1, ReGenX performs a comprehensive static analysis of the monolith, examining the codebase at multiple levels to construct an accurate, deeply detailed model of the application.

This analysis captures the fundamental building blocks of the system. It identifies classes and methods, measures structural and logical complexity, and maps how components depend on one another. Along the way, the platform records how different parts of the application interact with underlying databases—tracking reads, writes, and patterns of access that reveal how the monolith actually behaves in production-like scenarios.

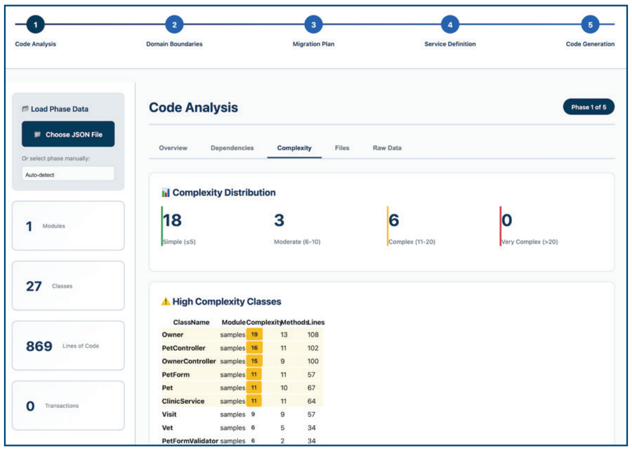
As part of this process, the platform pinpoints key elements such as:

- Classes and their associated responsibilities
- Dependencies each component relies on
- Data entities touched by each part of the system
- Transaction boundaries revealed through related database operations and transactional behaviors

This rich context becomes essential in later phases, especially when determining how to define service boundaries and evaluate extraction strategies without disrupting core workflows.

The outcome of Phase 1 is a comprehensive JSON representation of the monolith's structure and behavior. This artifact includes module breakdowns, dependency graphs, data access mappings, and detailed transaction boundary information. Within ReGenX, it serves as the authoritative source of truth that all downstream analysis and planning builds upon.

Phase 2: Domain Analysis



Code Analysis

1 Modules
27 Classes
869 Lines of Code
0 Transactions

Complexity Distribution

18 Simple (≤5)
3 Moderate (6-10)
6 Complex (11-20)
0 Very Complex (>20)

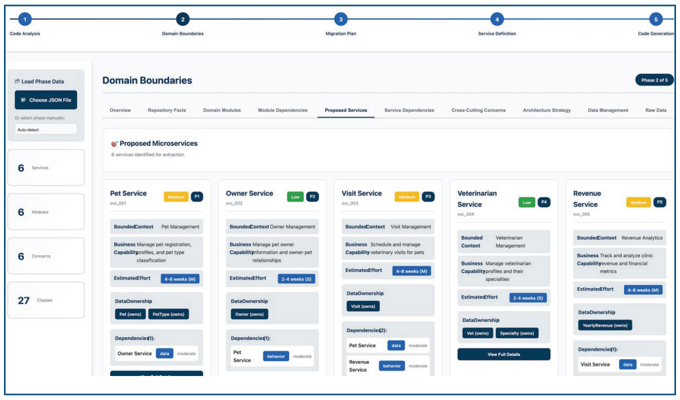
High Complexity Classes

ClassName	Module	Complexity	Methods	Lines
Owner	samples	19	13	108
PetController	samples	16	11	102
OwnerController	samples	15	9	100
PetForm	samples	11	11	57
Pet	samples	11	10	67
ClinicService	samples	11	11	64
Visit	samples	9	9	57
Vet	samples	6	5	34
PetFormValidator	samples	6	2	34

Figure 2. Representation of ReGenX Code Analysis Outcome

After the monolith's internal structure has been mapped, the next step is understanding how those technical components relate to real business domains. In Phase 2, ReGenX analyzes the system's behavior to uncover natural functional groupings and the domain concepts they represent.

As ReGenX processes these relationships, it



Domain Boundaries

6 Services
6 Modules
6 Concerns
27 Classes

Proposed Microservices
6 services identified for extraction

Pet Service — Medium — P1
svc_001
BoundedContext: Pet Management
Business Capability: Manage pet registration, profiles, and pet type classification
EstimatedEffort: 4-8 weeks (M)
DataOwnership: Pet (owns), PetType (owns)
Dependencies(1): Owner Service — data — moderate

Owner Service — Low — P2
svc_002
BoundedContext: Owner Management
Business Capability: Manage pet owner information and owner-pet relationships
EstimatedEffort: 2-4 weeks (S)
DataOwnership: Owner (owns)
Dependencies(1): Pet Service — behavior — moderate

Visit Service — Medium — P3
svc_003
BoundedContext: Visit Management
Business Capability: Schedule and manage veterinary visits for pets
EstimatedEffort: 4-8 weeks (M)
DataOwnership: Visit (owns)
Dependencies(2): Pet Service — data — moderate; Revenue Service — behavior — moderate

Veterinarian Service — Low — P4
svc_004
Bounded Context: Veterinarian Management
Business Capability: Manage veterinarian profiles and their specialties
EstimatedEffort: 2-4 weeks (S)
DataOwnership: Vet (owns), Specialty (owns)

Revenue Service — Medium — P5
svc_005
BoundedContext: Revenue Analytics
Business Capability: Track and analyze clinic revenue and financial metrics
EstimatedEffort: 4-8 weeks (M)
DataOwnership: YearlyRevenue (owns)
Dependencies(1): Visit Service — data — moderate

Figure 3. Example Output of ReGenX Domain Boundaries Phase

identifies patterns that reveal emerging domain boundaries. These patterns come from how components interact, which data entities they share, and how often certain parts of the system work together to support a specific capability. Through this analysis, the platform highlights:

- Cohesive groups of classes and methods
- Shared data entities
- Recurring collaboration patterns
- Areas where responsibilities align with distinct business concepts

These insights help teams see where the monolith naturally aligns with domain-driven design (DDD) principles and where seams already exist. This understanding becomes critical in later phases, informing how service boundaries should be drawn and where the cleanest separations can be made.

The output of Phase 2 is a structured set of domain clusters and conceptual groupings derived directly from the system's observed behavior. Inside ReGenX, this becomes a foundational guide for defining future services and evaluating extraction strategies based on real domain relationships rather than assumptions or guesswork.

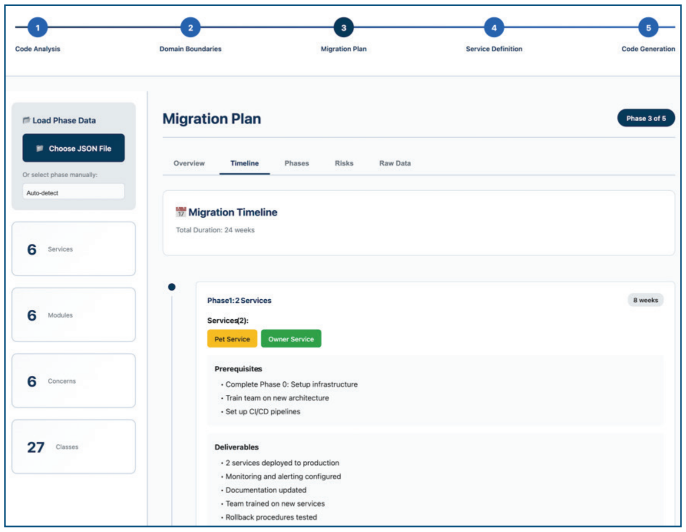
Phase 3: Migration Planning

Knowing the desired service boundaries is only the starting point. The difficult part is determining how to reach that future state without disrupting business operations. Some services are deeply entangled with others; some support mission critical workflows, and others can be modified with relatively little risk.

In Phase 3, ReGenX turns these domain insights into a practical, execution-ready migration plan. The platform examines a range of factors—service dependencies, extraction complexity, and business criticality—to shape a phased rollout that balances speed with safety.

As part of this planning process, the platform identifies:

- Which services must precede others in the migration sequence
- Where teams can safely work in parallel
- Which areas pose the highest migration risk
- Implement proven patterns—such as the Strangler Fig approach— to gradually shift functionality from a monolith into new, decoupled services



Migration Plan

6 Services
6 Modules
6 Concerns
27 Classes

Migration Timeline
Total Duration: 24 weeks

Phase1: 2 Services — 8 weeks
Services(2): Pet Service, Owner Service

Prerequisites
- Complete Phase 0: Setup infrastructure
- Train team on new architecture
- Set up CI/CD pipelines

Deliverables
- 2 services deployed to production
- Monitoring and alerting configured
- Documentation updated
- Team trained on new services
- Rollback procedures tested

Figure 4. Example Output From ReGenX Migration Plan

The result is a detailed migration roadmap that includes:

- The recommended order of service extraction
- Estimated timelines derived from complexity analysis
- Required prerequisites, including infrastructure, or architectural updates
- Testing, validation, and rollback strategies tailored to each phase

Teams can refine and adapt to this plan as they progress. Because the platform captures all domain and migration data in structured form, any updates to service boundaries or dependencies can be seamlessly incorporated—without rebuilding the plan from the ground up.

Phase 4: Service Definition

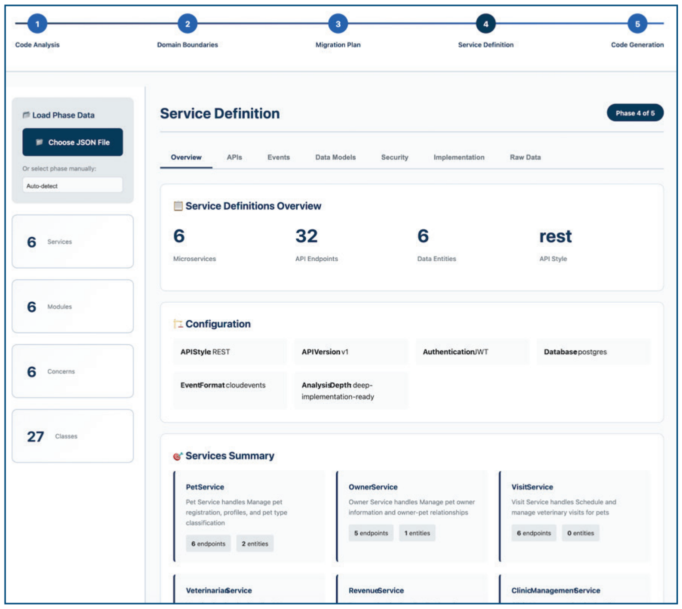
Many approaches to microservices stop once high-level service boundaries are defined, but teams still face the challenge of determining how APIs should be shaped, which events to publish, how data models should evolve, and what security rules to apply.

Phase 4 in ReGenX goes deeper. It uses the agreed upon domain boundaries and the migration plan to generate detailed, production-ready specifications for every microservice. For each service, the platform designs RESTstyle endpoints with the appropriate HTTP methods, request and response schemas, and error-handling patterns. It derives data models from the entities the service owns and produces database schema recommendations with sensible constraints and indexes.

As part of this process, the platform defines the key architectural elements for each service, including:

- REST endpoints, payload structures, and standardized error responses
- Data models and corresponding database schema suggestions
- Asynchronous events for cross service communication aligned with patterns such as CloudEvents
- Security requirements covering authentication, authorization rules, and data protection considerations

The level of specificity matters. Instead of offering vague instructions like "add a pet registration endpoint," the specification outlines the exact payload structures, validation rules, expected responses, error conditions, and illustrative interactions. It also ties these behaviors back to the underlying domain entities and the state transitions that govern their lifecycle.



Service Definition

6 Services
6 Modules
6 Concerns
27 Classes

Service Definitions Overview

6 Microservices
32 API Endpoints
6 Data Entities
rest API Style

Configuration

APIStyle REST
APIVersion v1
Authentication JWT
Database postgres
EventFormat cloudevents
AnalysisDepth deep-implementation-ready

Services Summary

PetService
Pet Service handles Manage pet registration, profiles, and pet type classification
6 endpoints 2 entities

OwnerService
Owner Service handles Manage pet owner information and owner-pet relationships
5 endpoints 1 entities

VisitService
Visit Service handles Schedule and manage veterinary visits for pets
6 endpoints 0 entities

VeterinarianService
RevenueService
ClinicManagementService

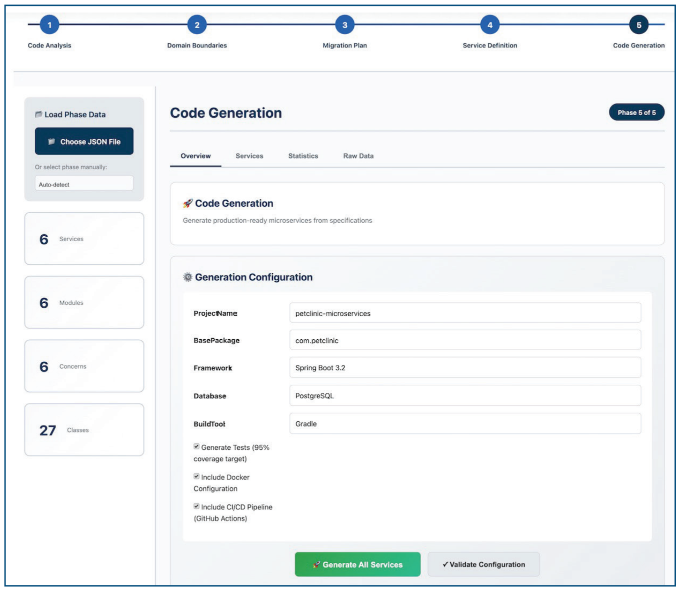
Figure 5. ReGenX Platform Output Example from Service Definition Phase

Phase 5: Code Generation

The final phase turns specifications into running code. This is where AI assistance reaches its most visible form.

In Phase 5, ReGenX generates complete service implementations from the service definitions produced earlier. It creates REST controllers or equivalent endpoints, business logic layers, data access components, event publishers and subscribers, configuration files, and basic observability instrumentation. The generated code follows production ready patterns. It includes structured error handling, input validation, transaction management, message serialization and deserialization, security middleware and health check endpoints.

Different technology stacks receive idiomatic implementations. Whether the target is Java with Spring Boot, Python with a modern web framework or Node.js with Express, the structure and patterns remain consistent while language specific conventions are respected. The result is not a throwaway prototype. Teams start with solid implementations that they can review, adapt to internal standards, and extend with additional business logic.



6 Services

6 Modules

6 Concerns

27 Classes

Code Generation

Code Generation

Generate production-ready microservices from specifications

Generation Configuration

ProjectName	petclinic-microservices
BasePackage	com.petclinic
Framework	Spring Boot 3.2
Database	PostgreSQL
BuildTool	Gradle

☑ Generate Tests (95% coverage target)

☑ Include Docker Configuration

☑ Include CI/CD Pipeline (GitHub Actions)

Figure 6. Example from ReGenX Phase 5 Code Configuration

How AI Integration Works in ReGenX

A key reason ReGenX is effective is how it incorporates AI into the phases where judgment and design matter most. The platform does more than simple pattern matching or template filling. It uses large language models (LLMs) that can reason about domains, interfaces, and code.

During domain discovery, AI applies DDD ideas to the data from code analysis. It looks for natural groupings of entities and behaviors that represent business capabilities rather than only technical structures. This allows it to detect domains that might not be obvious from file organization alone.

When defining services, AI uses knowledge of API design and microservices patterns to redesign monolith interactions for a distributed world. It chooses appropriate request and response styles, error handling strategies and event flows based on the analyzed behavior. The specifications reflect the real business logic of the system instead of generic templates.

For code generation, the AI produces code that is idiomatic for the chosen language. It uses familiar patterns and naming conventions, adds comments where needed, and creates hooks for testing.

The Human Element with ReGenX

Despite the level of automation, human expertise remains central and essential to successful system transformations and modernizations. ReGenX is designed as a partner for architects, engineers, and product leaders, not as a replacement for them. Architects review and refine the domain boundaries that the platform proposes. They bring knowledge about long-term business strategy, organizational constraints, and non-functional requirements that cannot be inferred from code alone.

Product managers and business stakeholders help prioritize which services to extract first. They weigh the potential value of new capabilities against the effort and risk of change. Engineers evaluate generated specifications and code. They ensure

“ Cadmus Logic.AI^{SM} ReGenX acts as a tireless analyst and assistant. It handles the heavy lifting of analysis and scaffolding, while people focus on decisions that require judgment and context.

that implementations align with team standards, integrate with existing infrastructure and meet performance expectations.

Operations and platform teams use the migration roadmap to plan infrastructure, monitoring, and roll out strategies. They make final decisions about deployment models, capacity and observability. In this model, ReGenX acts as a tireless analyst and assistant. It handles the heavy lifting of analysis and scaffolding, while people focus on decisions that require judgment and context.

> “By thoroughly analyzing dependencies and planning careful migration waves, organizations avoid many of the pitfalls that have caused past migrations to fail.

Practical Benefits of Using ReGenX

An AI-assisted, platform-based approach to microservices transformation offers several tangible benefits. Analysis that once required weeks of manual effort can now be completed in a much shorter period. Domain boundary proposals give teams a strong starting point for architecture discussions. Migration plans become more concrete and grounded in actual dependencies and complexity.

Service definitions capture implementation details early in the process, which reduces surprises during development. Generated code accelerates delivery by giving teams working implementations to refine rather than empty project skeletons.

Most importantly, the approach reduces risk. By thoroughly analyzing dependencies and planning careful migration waves, organizations avoid many of the pitfalls that have caused past migrations to fail.

Getting Started with ReGenX

Teams do not need to transform their entire monolith at once to benefit from the platform. A sensible approach is to start with a pilot project that focuses on a non-critical domain.

The pilot begins with running Phase 1 in ReGenX against the selected part of the system. The team reviews the analysis, then moves into Phase 2 to explore domain boundary proposals and refine them. Next, they use Phase 3 to design a migration plan for a small set of services, generate detailed service definitions in Phase 4, and produce code for one or two services in Phase 5.

This controlled experiment allows the organization to evaluate the quality of the analysis, the usefulness of the boundaries, the realism of the migration plan and the fidelity of the generated specifications and code. Successful pilots build confidence and create momentum for applying the platform to larger and more critical parts of the monolith.

Product Architecture Overview

Under the hood, ReGenX follows a modular architecture with clear separation between phases.

Talk to an Expert

See how we harness AI tosupport your microservices transformation.

Praveen Nedungottil, *Chief Technology Officer.*
Praveen.Nedungottil@cadmusgroup.com

Ram Polana, *Center of Excellence AI/ML Lead.*
Ramprasad.Polana@cadmusgroup.com

Each phase operates as an independent engine that consumes structured JSON input and produces structured JSON output. The data flow begins with code analysis, which outputs a JSON document describing modules, classes, dependencies, data access patterns, and transaction boundaries. The domain discovery engine consumes this document and produces a domain boundaries JSON that maps code to proposed services and records architectural guidance.

The migration planning engine uses the analysis and domain boundaries artifacts to create a migration plan JSON with phased timelines, service dependencies, risk assessments, and rollback strategies. The service definition engine brings together all earlier outputs to generate a service definitions JSON that contains API specifications, data models, event schemas, security configurations, and implementation patterns.

Finally, the code generation engine consumes the service definitions and produces source code organized by service, including controllers, business logic, data access, configuration, and tests. LLMs are integrated at the points where semantic understanding and design judgment are most valuable. Structured prompts provide context, constraints, and examples, which helps the AI produce consistent and high-quality output.

The platform remains language and framework agnostic at the analysis and definition stages. It works with concepts such as classes, methods, dependencies, and transactions that can be mapped across languages. Service definitions are expressed in technology neutral terms, which allows the generation engine to target different stacks without changing the earlier phases.

The modular design also supports extensibility and customization. For each client environment, we can add custom validation rules, integrate proprietary patterns, connect external tools, or adjust AI prompts to align output with internal standards.

Looking Forward

Microservices transformation will remain a complex endeavor. Distributed systems introduce their own challenges in reliability, observability, and operational overhead.

With Cadmus Logic.AI^{SM} ReGenX, we offer a way to approach this complexity with more structure and support. By combining a five-phase methodology with AI assistance and a unified platform, it helps teams move from monolithic legacy systems toward modern microservices architectures in a more predictable way.

The future of software architecture is about AI + architects, engineers, and product people working together, each contributing their strengths. Cadmus Logic.AII^{SM} ReGenX embodies this partnership by handling the heavy lifting of analysis and scaffolding so that people can focus on modeling the business, optimizing performance, and delivering value to users.