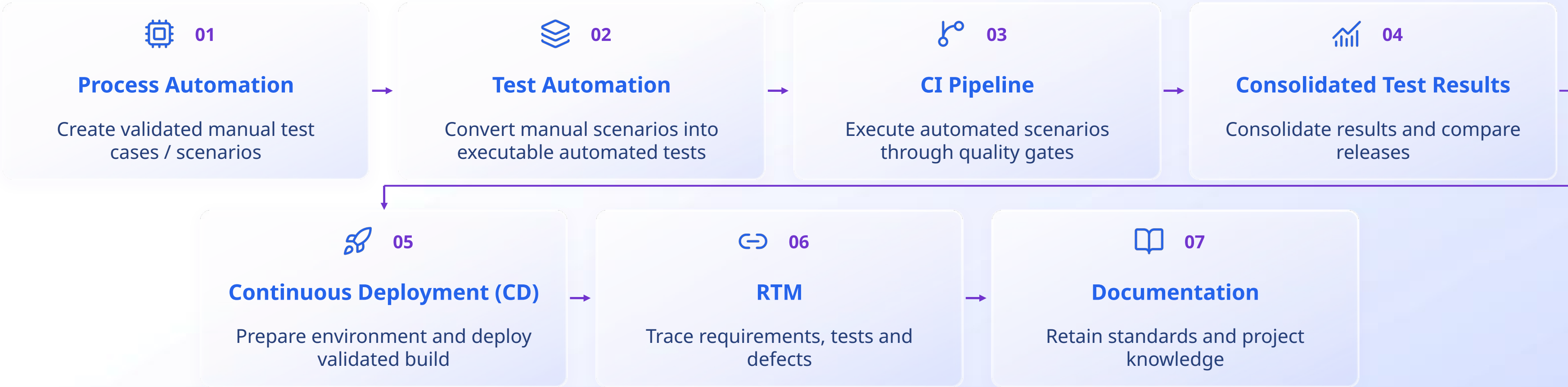


Responsible AI Intelligence-Driven Testing Strategy and Approach

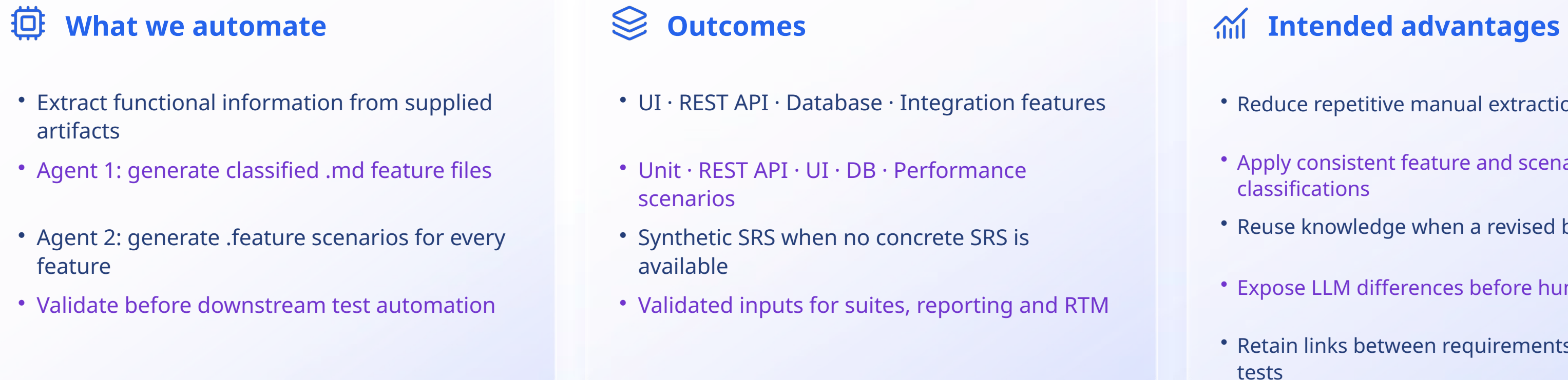
AI-driven testing strategy and approach for CAIR IV and V, with agentic AI tools for feature and scenario generation.

SEVEN-STAGE CONTROLLED LIFECYCLE



What Is Process Automation and Why?

Generate classified features and manual test scenarios from requirements or build artifacts, ready for validation and automation.



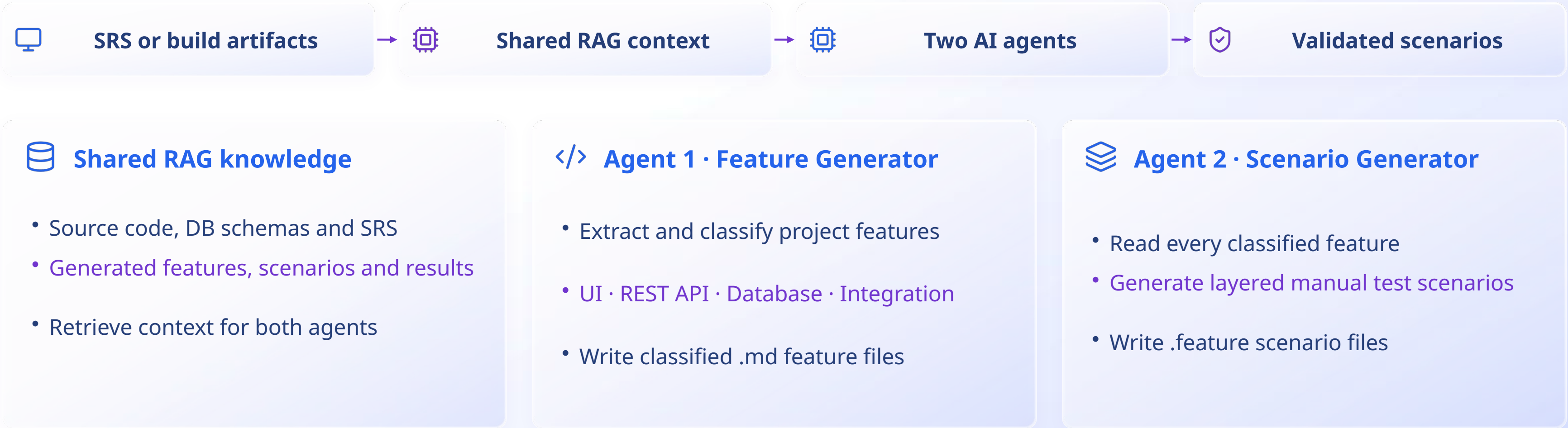
PROCESS AUTOMATION APPROACHES

- **Top-to-bottom: SRS → classified features → test scenarios**
- **Bottom-to-top: source code + DB schema → features / scenarios → synthetic SRS**

Delivered through an agentic AI framework: two purpose-built agents supported by shared RAG context.

Agentic AI Framework — Two Agents, Shared RAG

Shared project knowledge supports feature generation, scenario generation and validated reuse.



Parallel LLM outputs are reconciled, then manually and stakeholder validated before downstream automation.

RAG — Shared, Reusable Project Knowledge

INGEST SOURCE ARTIFACTS



GENERATED OUTPUTS RETURN TO RAG



Versioned artifacts, generated scenarios and execution evidence return to RAG for retrieval and validated reuse.

RAG Advantages — Retrieve, Reuse & Extend

Relevant evidence

- Retrieve front-end / back-end sources, DB schemas and SRS
- Find features and scenarios with source references
- Retrieve passed / failed test results by build and release

Create from context

- Draft new scenarios for selected features
- Derive user stories and workflows from stored evidence
- Review and validate generated outputs before reuse

Connected change context

- Compare source, features and tests across versions
- Reuse context when a revised build arrives
- Trace failures and changes back to source files

Versioned RAG context

Retrieve evidence

Generate / update scenarios

Validate + return to RAG

Intended advantages: less repeated discovery, reusable evidence and clearer traceability; no measured gains claimed.

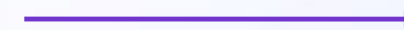
Two AI Agents — Features to Classified Scenarios

 SRS or revised build artifacts + relevant RAG context



Agent 1 · Feature Generator

- Extract and classify features
- UI layer · REST API layer
- Database layer · Integration layer
- Output: one .md file per feature



**Classified .md
features**

Agent 2 · Scenario Generator

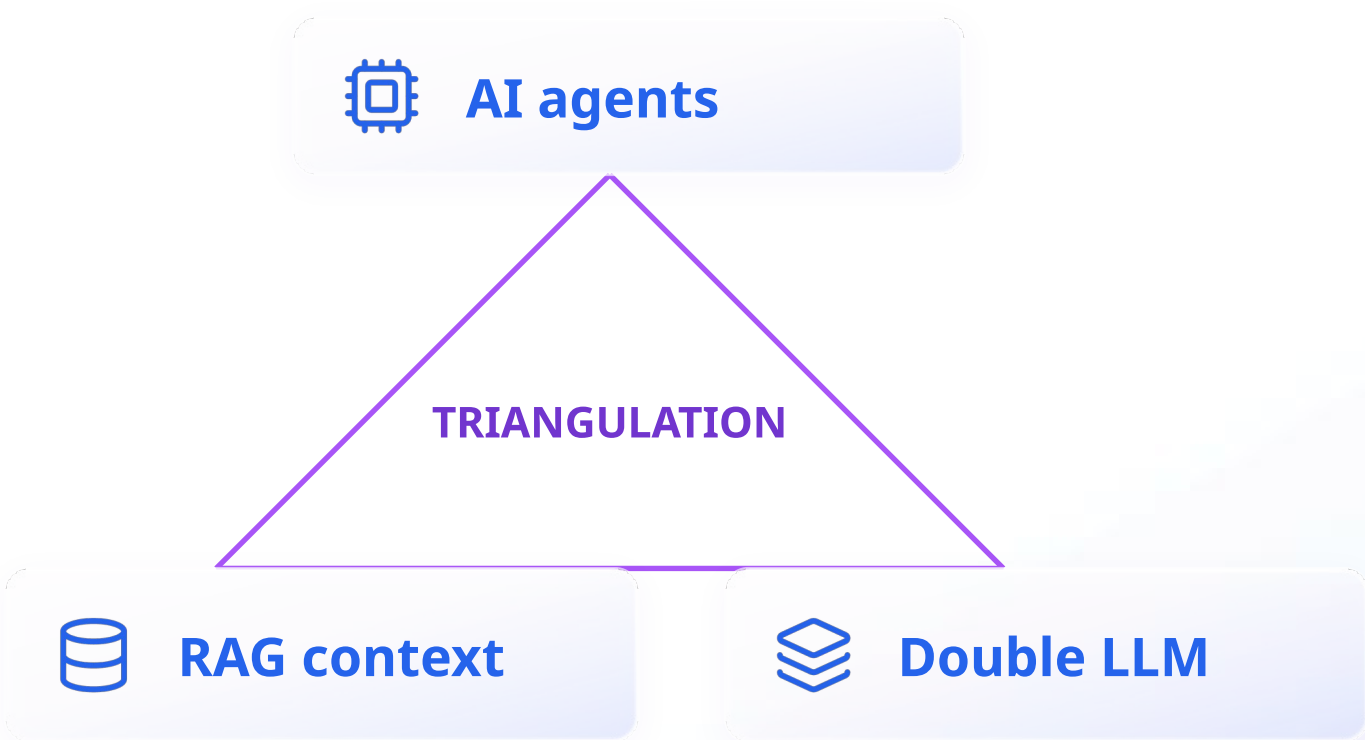
- Read every generated .md feature
- Generate test cases for each feature
- Unit · REST API · UI · DB · Performance
- Output: .feature files in respective folders

 Both agents write generated .md features and .feature scenarios back to RAG.

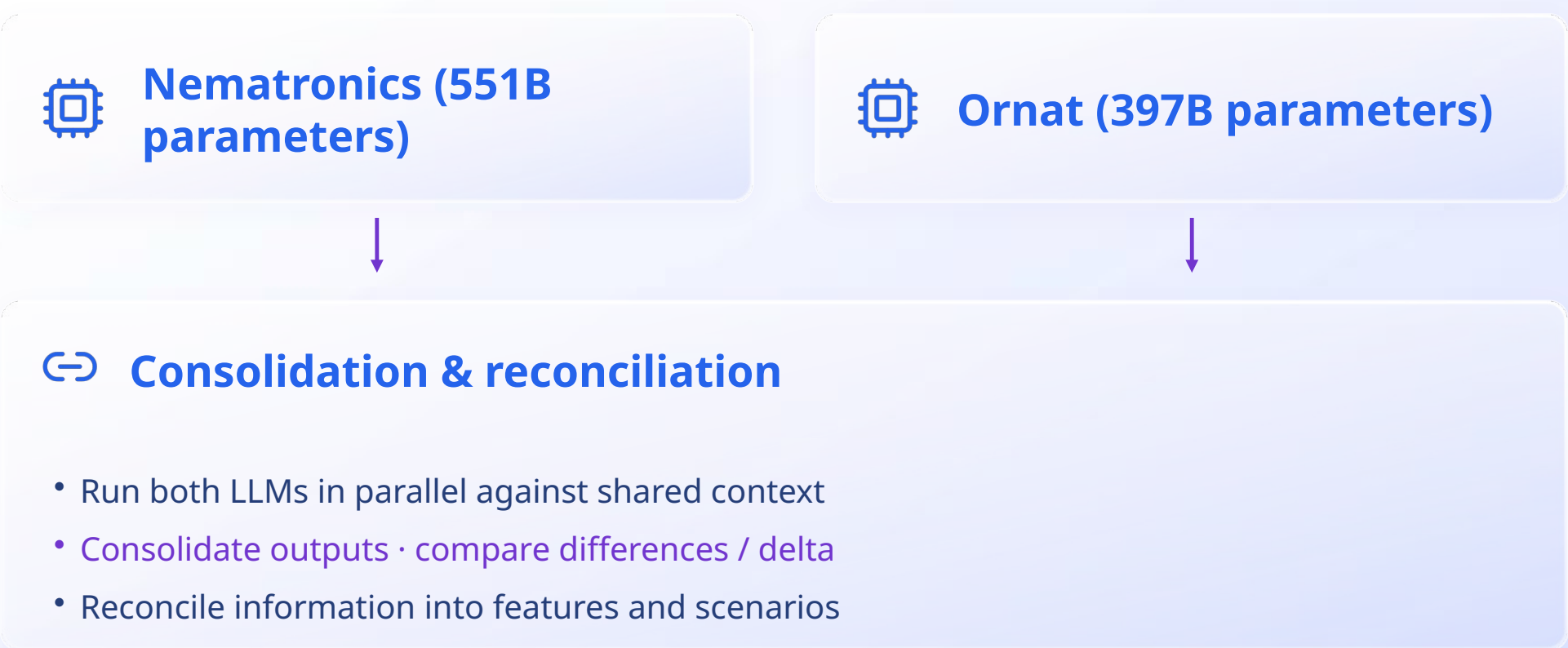
Revised build → updated context → classified features → per-feature scenarios → validation

Triangulation — Reconcile, Then Validate

AI · RAG · DOUBLE LLM



PARALLEL LLM OUTPUTS



HUMAN VALIDATION · SEQUENTIAL GATES



AI + RAG context validation and double-LLM delta validation precede manual and stakeholder validation.

Confidence Assessment / Confidence % — [to be assessed]

Top-to-Bottom Approach

Start with the SRS; generate classified feature files, then layered scenarios for every feature.



</> Feature Classification

- Classify features: UI · REST API · Database · Integration
- Generate one .md feature file for each classified feature

Scenario Classification

- Read each .md feature file and generate its test scenarios
- Unit · REST API · UI · DB · Performance scenarios
- Store .feature files in the respective layer folders

 Shared RAG layer: SRS, sources, features and scenarios provide context to both agents and both LLMs.



Confidence Assessment / Confidence % — [to be assessed]

Process Automation — Bottom-to-Top Approach (Reverse Engineering)

No concrete SRS available → derive a synthetic SRS using bottom-to-top reverse engineering.



</> Feature Generator

- Read front-end / back-end source and DB schema
- Classify UI · REST API · Database · Integration
- Generate one .md file for each feature

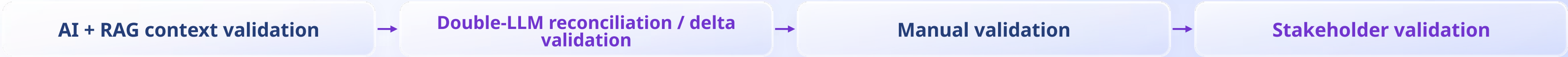
Scenario Generator

- Read each classified .md feature file
- Generate Unit · REST API · UI · DB · Performance scenarios for every feature
- Store .feature files in respective layer folders

RAG + Agentic AI / Double LLM

- Shared RAG supports both agents and LLMs
- Nematronics 551B + Ornat 397B run in parallel
- Consolidate outputs, compare deltas and reconcile

Triangulation Method / Validation



Confidence Assessment / Confidence % — [to be assessed]

Validated Features + Layered Scenarios → Synthetic SRS

TEST AUTOMATION

Test Automation — From Scenarios to Executable Automation

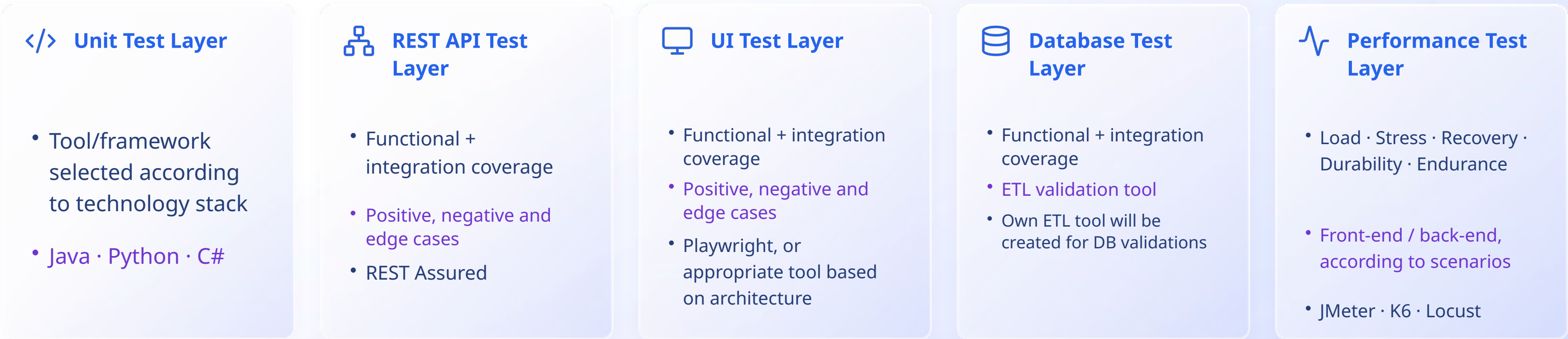
VALIDATED SCENARIOS → TEST SUITE CLASSIFICATION



Unit testing

Functional & integration testing

Non-functional testing



Manual test cases → executable smoke / regression suites → CI execution

TEST AUTOMATION · INTENDED ADVANTAGES

Test Automation — Quick Smoke Checks & Faster Regression

Quick smoke checks

- Run reusable smoke checks on each revised build
- Catch critical failures before wider regression
- Report failed checks promptly for correction

Faster regression cycles

- Repeat high / low regression suites across cycles
- Reduce repeated manual execution with reusable suites
- Rerun affected suites after fixes with traceable evidence

Generation to execution

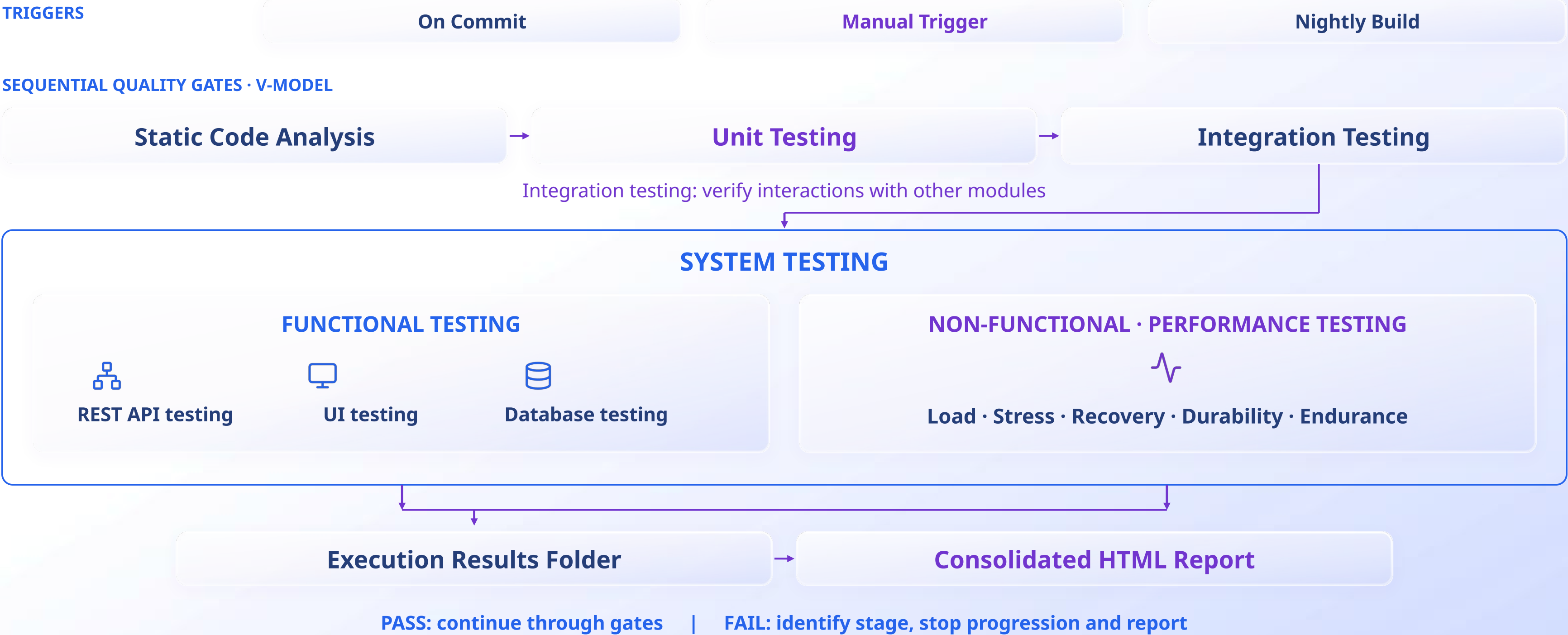
- Process automation creates validated manual cases
- Automate cases and retain reusable executable suites
- Keep suites in CI for consistent repeat execution

**New build / test cycle****Quick smoke checks****High / low regression****Failure feedback + targeted rerun**

Expected speed and effort benefits depend on coverage, environment and suite runtime; no timings or percentages are asserted.

CONTINUOUS INTEGRATION

CI Pipeline — V-Model Testing & Quality Gates



CONTINUOUS INTEGRATION · INTENDED ADVANTAGES

CI Pipeline — Advantages & Controlled Feedback

Repeatable execution

- Commit, manual and nightly triggers run defined gates
- Static code analysis → unit → integration → system
- Use consistent suite selection and result capture
- Check static code analysis against SonarQube security guidelines

Controlled progression

- Pass: continue to the next quality gate
- Fail: identify the stage and stop promotion
- Report failures for correction and verification
- Required security baseline: meet or exceed IMSAS security guidelines

Release evidence

- Retain results by build, release and test cycle
- Feed results into comparison reports
- Return results and traceable test evidence to RAG



Trigger



Execute quality gates



Capture + report results



Correct / rerun or promote

Security criteria require verification against SonarQube and IMSAS guidelines; compliance is not asserted.

EXECUTION & REPORTING

Consolidated Test Results — One View, Complete Traceability



LAYER-WISE REPORTS



CONSOLIDATED HTML REPORT · DASHBOARD CONCEPT

Module Name	Feature Name	.md Feature File	Unit Test Files Generated	Number of Unit Test Files	REST API Result	UI Result	DB Result	Performance Result
[module]	[feature]	[file.md]	[files]	[count]	[status]	[status]	[status]	[status]

DEEP DRILL-DOWN



Capture Release 1 / Release 2 / Release 3 with build, cycle and test identifiers.

Release-tagged results → module / feature / test drill-down → error, source file and line

Release Comparison — Results, Changes & Delta



Comparison evidence	Release 1	Release 2	Release 3
Build / cycle	[build / cycle]	[build / cycle]	[build / cycle]
Passed / failed / skipped	[results]	[results]	[results]
Features / tests / source files	[versioned links]	[versioned links]	[versioned links]

 Release 2 delta vs Release 1

- Identify added, modified and removed features / tests
- Show changed pass / fail outcomes and newly failing tests
- Link changes to source files, defects and execution evidence

 Release 3 + RAG feedback

- Compare R3 with R2 using the same release-linked evidence
- Retain reports, outcomes and source references in RAG
- Retrieve context to create or update scenarios, then validate

Report design only: bracketed fields require actual release data; no execution outcomes are invented.

CONTINUOUS DEPLOYMENT

Continuous Deployment — Automated Environment Preparation & Deployment

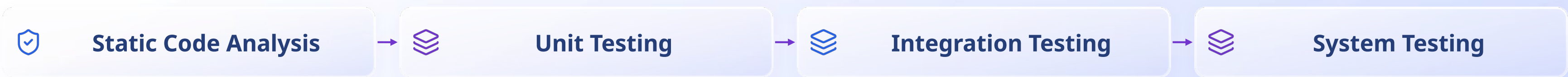
DEPLOYMENT FLOW



JAVA EXAMPLE



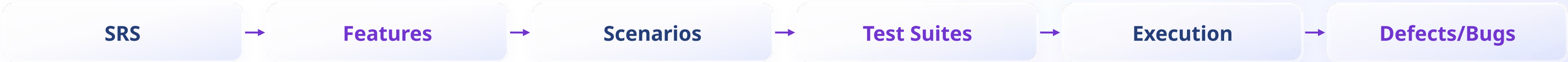
AFTER DEPLOYMENT · EXECUTE CI TEST FLOW



System testing: functional REST API / UI / DB + non-functional performance (load, stress, recovery, durability, endurance).

RTM — Bidirectional Requirement-to-Test Traceability

TOP-TO-BOTTOM



Backward trace: Bug → Scenario → Feature → SRS

BOTTOM-TO-TOP · REVERSE ENGINEERING



Backward trace: Bug → Scenario → Feature → Synthetic SRS

IBM ELM

Project-Level Tracking · Task-Level Tracking

- Automatically add every generated scenario to RTM, linked to its requirement and feature.
- Automatically add execution results to RTM, linked to the scenario, suite, build and defects.

Bidirectional links connect requirements, generated scenarios, execution evidence and defects.

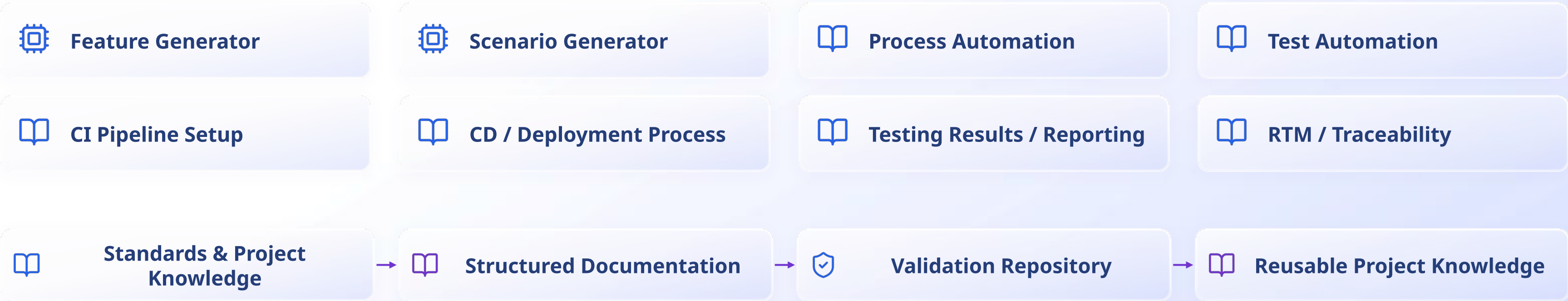
DOCUMENTATION & GOVERNANCE

Documentation — Standardized Knowledge Repository

REFERENCE STANDARDS



DOCUMENTATION COVERS



The complete strategy combines process automation, AI-assisted scenario generation, test automation, execution reporting, CI/CD, bidirectional traceability, and standardized documentation into one controlled lifecycle.

ONE CONTROLLED LIFECYCLE

Conclusion & Thank You

Thank you

CAIR · IV and V Project

Validated AI-assisted generation

- RAG context · Double-LLM reconciliation
- Manual validation · Stakeholder validation

Automation & quality gates

- Static code analysis · Unit · Integration · System
- Functional REST API / UI / DB · Non-functional performance
- CI execution → consolidated results → CD

Traceability & knowledge

- Bidirectional RTM · IBM ELM
- ISO Standards · IEEE Standards · DGSD

SRS OR SOURCE CODE → ONE CONNECTED STRATEGY



One connected strategy: process automation, testing, reporting, CI/CD, traceability and standardized documentation.