VALIDTEC.NET | TRACK 1

Agentic QA Systems & Orchestration

Implementation Playbook

20 ValidTec articles organized into five practical chapters - four articles per chapter - for designing, connecting, operating, and governing AI-assisted software testing workflows.

Track 1 focus

Design testable agent workflows. Add orchestration and evaluation gates. Instrument agent activity so outcomes can be reviewed, repeated, and improved.

Prepared for ValidTec.net
Version 1.0 - August 2026

About This Playbook

This book is a practical implementation companion for ValidTec Track 1: Agentic QA Systems & Orchestration. It organizes 20 Track 1 article topics into five stages that a software testing engineer can execute in sequence.

> **Important note**
> The chapters synthesize the themes represented by the source article titles into an original implementation framework. Readers should use the original ValidTec articles for the full article-specific discussion, examples, and references.

How to use it

1. Choose one real QA workflow with a measurable pain point.
2. Complete Chapter 1 to define the agent mission, boundaries, and success measures.
3. Complete Chapters 2 and 3 to design orchestration, context, protocols, and tool access.
4. Use Chapter 4 to establish the tester-AI operating workflow and implementation environment.
5. Use Chapter 5 to add observability, evaluation gates, governance, and auditability.
6. Run the 90-day roadmap at the end of the book and keep the artifacts as portfolio evidence.

Track 1 completion standard

- A workflow is decomposed into explicit agent or human responsibilities.
- Inputs, context, tools, permissions, outputs, and handoffs are documented.
- Critical outputs pass deterministic checks or human review before downstream action.
- Runs are observable enough to reconstruct what happened and why.
- The workflow has measurable quality, cost, speed, and reliability targets.
- The design can be tested repeatedly in CI, staging, or a controlled QA environment.

Contents

CHAPTER 1

Define the Agentic QA Mission

Start with the problem, agent roles, risk, and a testable MVP.

Chapter outcome

A one-page Agentic QA Charter that names the workflow, users, agent roles, boundaries, success metrics, and first implementation slice.

This chapter converts four related Track 1 articles into one implementation playbook. Complete the exit criteria before moving to the next chapter.

The four source articles

#	Source article	Role in this chapter
1	**Agentic AI for Software Testing: A Practical Playbook to Cut Costs-Fast**	Business case, scope, cost and speed targets.
2	**6 Types of AI Agents for Software Testing - Benefits & How to Use Each One**	Select agent responsibilities instead of starting from tools.
3	**LLMs are everywhere-here's a practical test strategy startups and enterprises can adopt today**	Define the test strategy and risk coverage around AI-assisted behavior.
4	**Building Agentic AI Projects as a Software Testing Engineer**	Turn the idea into an implementable QA project and portfolio artifact.

Four-phase playbook

Phase	Action	Output	Gate
1. Problem	Pick one repetitive, evidence-rich QA workflow.	Problem statement + baseline.	Pain is measurable.
2. Roles	Map human and agent responsibilities.	Responsibility matrix.	No ambiguous owner.
3. Controls	Define risks, approvals, and prohibited actions.	Control list.	High-risk actions gated.
4. MVP	Choose the smallest useful end-to-end slice.	Agentic QA Charter.	Can be tested in 2-4 weeks.

Operating principles

- Automate a bounded workflow before attempting broad autonomy.
- Assign agents to explicit responsibilities such as planning, retrieval, execution, validation, or reporting.
- Measure value with QA outcomes: defect detection, coverage, cycle time, rework, cost per run, and release confidence.
- Keep a human approval point wherever a wrong action could alter production data, code, access, or customer outcomes.

Implementation steps

1. Choose the workflow

Examples: defect triage, test design from requirements, API regression analysis, production-issue reproduction, or release evidence summarization.

2. Establish a baseline

Capture current execution time, manual touches, recurring defects, escape rate, and cost or effort. Without a baseline, productivity claims cannot be validated.

3. Define agent roles

Use role names that express responsibility: Planner, Retriever, Test Designer, Executor, Validator, Reporter. One agent may perform several roles initially.

4. Set autonomy boundaries

List what the workflow may read, propose, execute, modify, and publish. Require explicit approval for destructive or externally visible actions.

5. Define success metrics

Select 3-5 measures. Recommended: task completion rate, validator pass rate, false-positive rate, human correction rate, run cost, and cycle time.

6. Select the MVP

Build one thin vertical slice that produces a useful artifact even when a human remains in the loop.

Artifacts to keep

- Agentic QA Charter
- Role and responsibility matrix
- Baseline measurement sheet
- Risk and approval register
- MVP acceptance criteria

Exit criteria

[ ] The workflow has one named owner and one defined user.

[ ] The agent responsibilities are explicit and non-overlapping.

[ ] Success metrics have current baseline values or a plan to collect them.

[ ] High-risk actions are blocked or require approval.

[ ] The MVP can be executed repeatedly against known test inputs.

Chapter completion rule

Do not advance because the design looks impressive. Advance when the artifacts exist, the exit criteria can be demonstrated, and the workflow can be executed against known cases.

Implementation worksheet

Capture the decisions for this chapter before implementation moves forward.

Workflow / use case	
Primary owner and reviewer	
Metric + target	
Largest risk + control	
Evidence / artifact to retain	

CHAPTER 2

Design the Orchestrated QA Workflow

Combine context, models, workflow patterns, and automation into a controlled execution path.

Chapter outcome

A workflow blueprint showing stages, handoffs, state, model choices, retrieval, deterministic checks, retries, and failure handling.

This chapter converts four related Track 1 articles into one implementation playbook. Complete the exit criteria before moving to the next chapter.

The four source articles

#	Source article	Role in this chapter
1	**Advancing Software Testing with AI Orchestration: RAG, Prompt Engineering, and SLM Integration**	Architecture for grounded context, prompt control, and model routing.
2	**Hybrid Approach for Impressive Results**	Combine deterministic automation, human review, and model-driven reasoning.
3	**Workflow Patterns That Supercharge Software Test Engineering**	Choose reusable orchestration patterns for common QA tasks.
4	**n8n for Software Testing: Orchestrate, Observe, and Automate Your QA**	Implement visible workflow steps, integrations, triggers, and operational automation.

Four-phase playbook

Phase	Action	Output	Gate
1. Decompose	Break the MVP into ordered stages.	Workflow map.	Each stage has one purpose.
2. Route	Choose code, SLM/LLM, retrieval, or human for each stage.	Execution matrix.	Cheapest reliable option chosen.
3. Guard	Add schemas, validators, retries, and stop conditions.	Control flow.	Bad output cannot silently pass.

4. Observe	Add run IDs, timing, cost, and step status.	Traceable workflow.	Failures are diagnosable.

Operating principles

- Use deterministic code for rules that can be expressed deterministically; reserve models for ambiguity, synthesis, and reasoning.
- Ground model steps with approved context instead of relying on memory or broad prompts.
- Make handoffs explicit: every step should have a defined input contract, output contract, timeout, retry policy, and failure path.
- Separate orchestration from evaluation so a workflow cannot mark itself successful merely because it completed.

Implementation steps

1. Draw the happy path

Represent the workflow as Trigger -> Gather Context -> Plan -> Execute -> Validate -> Report. Add branches only after the main path is understandable.

2. Classify every step

Mark each step as deterministic code, retrieval, model reasoning, tool call, human decision, or evaluation gate.

3. Choose the model mix

Use smaller or specialized models where the task is narrow and measurable; use larger models only where the quality gain justifies cost and latency.

4. Add state and correlation

Give each run a unique ID. Persist important inputs, tool results, decisions, validator results, and final outputs under that run.

5. Design retries deliberately

Retry only transient failures. For semantic failures, change the prompt/context or route to review rather than repeating the same call indefinitely.

6. Add a dead-end path

Define what happens after validation failure, missing context, unavailable tools, exceeded cost, or repeated retries.

Artifacts to keep

- Workflow sequence diagram
- Step classification matrix
- Model and routing matrix
- Retry/timeout policy
- Run-state schema

Exit criteria

[ ] Every model step is grounded by defined inputs or retrieval.

[ ] Every tool call has a timeout and failure path.

[ ] Validators are separate from the step they evaluate.

[ ] Run IDs correlate model calls, tools, validations, and outputs.

[ ] A failed run stops safely and produces actionable diagnostics.

Chapter completion rule

Do not advance because the design looks impressive. Advance when the artifacts exist, the exit criteria can be demonstrated, and the workflow can be executed against known cases.

Implementation worksheet

Capture the decisions for this chapter before implementation moves forward.

Workflow / use case	
Primary owner and reviewer	
Metric + target	
Largest risk + control	
Evidence / artifact to retain	

CHAPTER 3

Connect Context, Skills, and Tools

Give agents useful context and capabilities without creating an uncontrolled integration surface.

Chapter outcome

A Tool and Context Contract that defines approved sources, skills, protocols, credentials, permissions, tool schemas, and grounding evidence.

This chapter converts four related Track 1 articles into one implementation playbook. Complete the exit criteria before moving to the next chapter.

The four source articles

#	Source article	Role in this chapter
1	**How URL Context Grounding Supercharges Testing Agents**	Ground test decisions in selected live or published sources.
2	**How Anthropic's Agent Skills Can Level-Up Software Testing**	Package reusable testing instructions and domain capabilities.
3	**ACP + MCP + A2A for QA: one simple, vendor-neutral blueprint (with pragmatic alternatives)**	Separate context/tool access and agent communication through explicit interfaces.
4	**From MCP Servers to CLI Armies: How Agents Really Scale Tooling (and Why Testing Must Evolve)**	Scale tool execution while preserving contracts, permissions, and testability.

Four-phase playbook

Phase	Action	Output	Gate
1. Context	Identify authoritative sources.	Source registry.	Source owner and freshness known.
2. Skills	Package repeatable QA instructions.	Skill catalog.	Inputs/outputs documented.
3. Tools	Expose approved APIs/CLI/actions.	Tool contract.	Least privilege enforced.
4. Protocols	Standardize exchanges and handoffs.	Interface map.	Contracts can be tested.

Operating principles

- Context is an input to test, version, and trace - not an invisible convenience.
- Expose the smallest tool surface that satisfies the workflow.
- Wrap tools behind stable schemas so the agent sees predictable names, arguments, results, and errors.
- Treat skills, prompts, protocols, CLIs, APIs, and connectors as versioned dependencies with regression coverage.

Implementation steps

1. Create a source registry

List requirement sources, issue trackers, repositories, test reports, logs, knowledge bases, and URLs. Record owner, trust level, freshness expectation, and permitted use.

2. Define grounding evidence

Require the workflow to preserve source identifiers or excerpts sufficient for a reviewer to verify why a test or decision was produced.

3. Create reusable skills

Package stable instructions such as generate API negative tests, summarize a run, map production defects to regression candidates, or validate release evidence.

4. Normalize tools

For every tool, document name, description, arguments, expected response, errors, timeout, authentication, and whether it reads or writes.

5. Apply least privilege

Use read-only credentials by default. Separate read tools from write tools. Require approval or narrow service accounts for changes.

6. Test the integration layer

Create contract tests for tool schemas, invalid arguments, missing credentials, rate limits, malformed responses, and unavailable dependencies.

Artifacts to keep

- Authoritative source registry
- Grounding evidence standard
- Reusable skill catalog
- Tool/API/CLI contract catalog
- Integration contract-test suite

Exit criteria

[ ] Every source has an owner and trust level.

[ ] Every write-capable tool is explicitly identified and controlled.

[ ] Tool arguments and outputs are schema-validated where practical.

[ ] Agent outputs retain enough evidence for source verification.

[ ] Integration failures are covered by repeatable tests.

Chapter completion rule

Do not advance because the design looks impressive. Advance when the artifacts exist, the exit criteria can be demonstrated, and the workflow can be executed against known cases.

Implementation worksheet

Capture the decisions for this chapter before implementation moves forward.

Workflow / use case	
Primary owner and reviewer	
Metric + target	
Largest risk + control	
Evidence / artifact to retain	

CHAPTER 4

Build the Tester-AI Operating Workflow

Turn agent concepts into a repeatable daily engineering system for test design, automation, execution, and reporting.

Chapter outcome

A working QA engineering environment with project instructions, reusable prompts/skills, source-control workflow, and one implemented agent workflow on a chosen platform.

This chapter converts four related Track 1 articles into one implementation playbook. Complete the exit criteria before moving to the next chapter.

The four source articles

#	Source article	Role in this chapter
1	**How Claude Cowork + Claude Code can 2-5x a Software Test Engineer's effectiveness**	Split collaborative analysis from repository-aware implementation work.
2	**How Claude Cowork + Claude Code can make a Software Test Engineer 10-20x faster (if you set up Preferences + CLAUDE.md + Skills + Prompts)**	Standardize persistent instructions, project context, skills, and reusable prompts.
3	**Prompting Beyond the Chatbot: A Better Productivity Strategy for Software Testing Engineers**	Treat prompts as workflow assets with inputs, outputs, checks, and context boundaries.
4	**Training for Microsoft Agent Framework + Azure AI Foundry Agent Service.**	Translate the operating model into a structured enterprise agent implementation path.

Four-phase playbook

Phase	Action	Output	Gate
1. Standardize	Create project instructions and reusable prompts.	AI workspace pack.	Team can reproduce setup.
2. Implement	Build one workflow on a branch or sandbox.	Working prototype.	Normal tests still pass.
3. Evaluate	Run golden cases and edge cases.	Evaluation report.	Quality threshold met.

4. Operationalize	Document triggers, ownership, and support.	Runbook.	Another engineer can operate it.

Operating principles

- Persistent project instructions should encode repository facts, commands, quality rules, and boundaries that should not be rediscovered every session.
- Prompts should be repeatable work instructions, not one-off conversational requests.
- Keep repository changes on branches with normal reviews, tests, and CI gates even when AI generates the change.
- Use a platform only after the workflow and contracts are clear; platforms accelerate an architecture but do not replace it.

Implementation steps

1. Create project instructions

Document repo map, setup commands, test commands, branch policy, code conventions, secrets rules, environment assumptions, and prohibited actions.

2. Create prompt assets

For each recurring task, define purpose, required context, input format, output format, quality checks, and when to escalate to a human.

3. Separate analysis from changes

Use conversational or collaborative tools for discovery and design; use repository-aware coding workflows for implementation with diffs, branches, tests, and review.

4. Build a golden task set

Select 10-30 representative cases with expected results. Include normal, boundary, missing-context, conflicting-context, tool-failure, and unsafe-action cases.

5. Implement on the chosen platform

Map the already-defined roles, tools, context, state, and gates into the chosen agent framework or service. Keep adapters thin so the workflow is not locked to one vendor.

6. Create an operator runbook

Document startup, inputs, expected outputs, common failures, rollback, logs, escalation, and how to disable automation safely.

Artifacts to keep

- Project instruction file / AI workspace guide
- Reusable prompt and skill library
- Golden task set
- Branch + CI workflow
- Agent operator runbook

Exit criteria

[ ] A second engineer can reproduce the environment from documentation.

[ ] Generated code or automation follows normal source-control and CI controls.

[ ] Golden cases achieve the agreed pass threshold.

[ ] The workflow handles missing or conflicting context explicitly.

[ ] The operator runbook includes disable/rollback and escalation steps.

Chapter completion rule

Do not advance because the design looks impressive. Advance when the artifacts exist, the exit criteria can be demonstrated, and the workflow can be executed against known cases.

Implementation worksheet

Capture the decisions for this chapter before implementation moves forward.

Workflow / use case	
Primary owner and reviewer	
Metric + target	
Largest risk + control	
Evidence / artifact to retain	

CHAPTER 5

Operate, Observe, and Govern Agents

Make agent behavior reviewable, measurable, auditable, and safe to evolve.

Chapter outcome

An Agent QA Scorecard and run-audit model that lets senior QA engineers trace decisions, detect regressions, control releases, and improve the workflow over time.

This chapter converts four related Track 1 articles into one implementation playbook. Complete the exit criteria before moving to the next chapter.

The four source articles

#	Source article	Role in this chapter
1	**Deepchecks "Know Your Agent" (KYA): The QA Upgrade for the Multi-Agent Era**	Adopt an explicit quality model for agent identity, capabilities, risks, and evaluation.
2	**Know your agent tools: how QA stays productive as Claude Code & Cowork scale**	Maintain visibility into changing tools, permissions, and execution modes.
3	**How Senior QA Engineers Monitor AI Agents End-to-End: Runs -> Tools -> Decisions -> Audit Trail**	Trace complete runs from input through tools, decisions, validators, and outputs.
4	**ADLC and the New Challenge for Software Testing in 2026 and Beyond**	Extend lifecycle quality practices to agent design, evaluation, release, and continuous monitoring.

Four-phase playbook

Phase	Action	Output	Gate
1. Inventory	Record agent, prompt, model, tools, permissions, owners.	Agent card.	Version is identifiable.
2. Trace	Capture run events and decisions.	Audit trail.	Run can be reconstructed.
3. Evaluate	Score quality and behavior.	Agent QA scorecard.	Release threshold enforced.
4. Improve	Analyze failures and update tests.	Regression backlog.	Escapes become future gates.

Operating principles

- If a run cannot be reconstructed, the system is not sufficiently testable for important workflows.
- Quality metrics must cover both task success and behavior: correctness, groundedness, tool selection, policy compliance, cost, latency, and stability.
- Agent versions, prompts, skills, models, tools, and policies should be traceable to each run.
- Changes to an agent workflow should pass regression gates just like changes to application code.

Implementation steps

1. Create an agent card

Record purpose, owner, version, model, system/project instructions, skills, tools, data sources, permissions, validators, known limitations, and escalation rules.

2. Define the run event model

Capture run ID, timestamps, input identifiers, context sources, prompt/skill version, model, tool calls, arguments or safe hashes, results, decisions, validator results, cost/usage, final output, and human intervention.

3. Build the scorecard

Recommended dimensions: task success, factual/grounded correctness, schema compliance, tool correctness, policy compliance, human correction rate, latency, cost, and repeatability.

4. Create release gates

Require regression evaluation before changing prompts, models, tools, skills, orchestration logic, or permission scope. Compare the candidate to a known baseline.

5. Feed production learning back

Convert incidents, customer issues, unexpected tool behavior, and human corrections into new golden cases, validators, and monitoring checks.

6. Establish review cadence

Review failures weekly during early rollout, then shift to a cadence based on risk and change rate. Keep owners accountable for unresolved regressions.

Artifacts to keep

- Agent card / KYA inventory
- Run event and audit schema
- Agent QA scorecard
- Regression evaluation suite
- Agent change and release checklist

Exit criteria

[ ] Every production-relevant run has a correlation ID.

[ ] Prompt/model/tool/skill versions are attributable to a run.

[ ] Release criteria include measurable evaluation thresholds.

[ ] Production failures are converted into regression coverage.

[ ] Permissions and write-capable tools are reviewed when the workflow changes.

Chapter completion rule

Do not advance because the design looks impressive. Advance when the artifacts exist, the exit criteria can be demonstrated, and the workflow can be executed against known cases.

Implementation worksheet

Capture the decisions for this chapter before implementation moves forward.

Workflow / use case	
Primary owner and reviewer	
Metric + target	
Largest risk + control	
Evidence / artifact to retain	

Track 1 Reference Architecture

Use this as a compact mental model for the complete playbook. Each layer corresponds to implementation work completed in one or more chapters.

5. Observe & Govern	Run trace | Agent card | Evaluation scorecard | Regression gate | Audit trail
4. Operate	Project instructions | Prompt/skill library | Source control | CI | Operator runbook
3. Connect	Grounded sources | Skills | APIs/CLI | Tool contracts | Permissions | Protocol adapters
2. Orchestrate	State | Routing | RAG | Model selection | Retries | Validators | Failure paths
1. Mission	Use case | Agent roles | Baseline | Success metrics | Risk boundaries | MVP

Golden-path run

1. Receive a bounded QA task and assign a run ID.
2. Retrieve only the approved context required for that task.
3. Plan the work using the defined agent/human responsibilities.
4. Invoke tools through explicit contracts and least-privilege credentials.
5. Validate intermediate and final outputs with deterministic checks, independent evaluators, or human approval.
6. Persist trace evidence, quality results, timing, and cost.
7. Publish the approved output or route the failed run to investigation.
8. Convert meaningful failures into new regression cases.

90-Day Track 1 Implementation Roadmap

Window	Focus	Work	Exit
Days 1-15	**Mission and baseline**	Choose one workflow; create Agentic QA Charter; collect baseline; define metrics and risk boundaries.	Charter approved; MVP inputs assembled.
Days 16-30	**Workflow architecture**	Decompose stages; select deterministic vs model steps; define state, validators, retries, and failure paths.	Blueprint can be walked through end-to-end.
Days 31-45	**Context and tools**	Register sources; create tool contracts; add least privilege; build integration contract tests.	Tools and context are testable independently.
Days 46-60	**Working prototype**	Implement the workflow; create project instructions, prompt/skill assets, branch workflow, and golden cases.	Prototype passes initial golden set.
Days 61-75	**Evaluation and observability**	Add run tracing, agent card, scorecard, cost/latency metrics, and change-regression gates.	Failures are diagnosable and comparable to baseline.
Days 76-90	**Controlled rollout**	Pilot with real QA work; retain human approvals; collect corrections; convert failures into regression coverage.	Documented go/no-go decision for broader use.

Minimum portfolio package

- Architecture diagram and Agentic QA Charter.
- Workflow configuration or code with a README.
- Prompt/skill assets and tool schemas with secrets removed.
- Golden task set and evaluation report.
- Run trace example showing tools, decisions, validators, and final status.
- Before/after metrics comparing the baseline workflow to the agent-assisted workflow.
- One retrospective explaining failures, changes made, and next regression tests.

Portfolio framing

Show engineering discipline, not just model output. Employers and reviewers should be able to see the problem, controls, repeatability, test evidence, and measured result.

Track 1 Article Index

All 20 source article titles supplied for this first Track 1 book, grouped four per implementation chapter.

Chapter	Article	Playbook stage
Chapter 1	Agentic AI for Software Testing: A Practical Playbook to Cut Costs-Fast	Define the Agentic QA Mission
Chapter 1	6 Types of AI Agents for Software Testing - Benefits & How to Use Each One	Define the Agentic QA Mission
Chapter 1	LLMs are everywhere-here's a practical test strategy startups and enterprises can adopt today	Define the Agentic QA Mission
Chapter 1	Building Agentic AI Projects as a Software Testing Engineer	Define the Agentic QA Mission
Chapter 2	Advancing Software Testing with AI Orchestration: RAG, Prompt Engineering, and SLM Integration	Design the Orchestrated QA Workflow
Chapter 2	Hybrid Approach for Impressive Results	Design the Orchestrated QA Workflow
Chapter 2	Workflow Patterns That Supercharge Software Test Engineering	Design the Orchestrated QA Workflow
Chapter 2	n8n for Software Testing: Orchestrate, Observe, and Automate Your QA	Design the Orchestrated QA Workflow
Chapter 3	How URL Context Grounding Supercharges Testing Agents	Connect Context, Skills, and Tools
Chapter 3	How Anthropic's Agent Skills Can Level-Up Software Testing	Connect Context, Skills, and Tools
Chapter 3	ACP + MCP + A2A for QA: one simple, vendor-neutral blueprint (with pragmatic alternatives)	Connect Context, Skills, and Tools
Chapter 3	From MCP Servers to CLI Armies: How Agents Really Scale Tooling (and Why Testing Must Evolve)	Connect Context, Skills, and Tools
Chapter 4	How Claude Cowork + Claude Code can 2-5x a Software Test Engineer's effectiveness	Build the Tester-AI Operating Workflow
Chapter 4	How Claude Cowork + Claude Code can make a Software Test Engineer 10-20x faster (if you set up Preferences + CLAUDE.md + Skills + Prompts)	Build the Tester-AI Operating Workflow
Chapter 4	Prompting Beyond the Chatbot: A Better Productivity Strategy for Software Testing Engineers	Build the Tester-AI Operating Workflow

Chapter 4	Training for Microsoft Agent Framework + Azure AI Foundry Agent Service.	Build the Tester-AI Operating Workflow
Chapter 5	Deepchecks "Know Your Agent" (KYA): The QA Upgrade for the Multi-Agent Era	Operate, Observe, and Govern Agents
Chapter 5	Know your agent tools: how QA stays productive as Claude Code & Cowork scale	Operate, Observe, and Govern Agents
Chapter 5	How Senior QA Engineers Monitor AI Agents End-to-End: Runs -> Tools -> Decisions -> Audit Trail	Operate, Observe, and Govern Agents
Chapter 5	ADLC and the New Challenge for Software Testing in 2026 and Beyond	Operate, Observe, and Govern Agents

What Completion Looks Like

Track 1 is complete when the reader can demonstrate one agentic QA workflow that is useful, bounded, testable, observable, and governed. The goal is not maximum autonomy. The goal is reliable quality engineering with AI-assisted execution.

The finished system

A real QA task enters through a defined trigger; receives approved context; follows an explicit orchestration path; calls only authorized tools; passes validation gates; produces an auditable result; and feeds failures back into regression coverage.

Next improvements

- Expand the golden set before expanding autonomy.
- Add new tools only when existing tools cannot satisfy the workflow reliably.
- Compare models with the same evaluation cases before changing the production route.
- Reduce token/cost waste after quality is stable, not before.
- Promote production incidents and human corrections into permanent tests.
- Revisit permissions and approval boundaries whenever the workflow gains new capabilities.

Continue with the source articles and future Track 1 updates at ValidTec.net.