AI-Driven QA Automation & Self-Improving Test Platform

PRODUCT REQUIREMENTS DOCUMENT (PRD)

(Note: Proprietary details have been modified for confidentiality)

Version: 2.1
Product Manager: Ben Sweet
Date: April 2024

Executive Summary

The organization's current QA processes rely heavily on manual tests, brittle automation scripts, inconsistent coverage, and multi-day regression cycles. These constraints slow delivery, increase defect leakage, elevate operational cost, and complicate compliance with federal oversight.

This initiative delivers a scalable **AI-Driven QA Automation & Self-Improving Test Platform** that automatically generates, executes, self-heals, and improves test coverage across programs. The platform accelerates release pipelines, reduces QA labor, strengthens compliance audit posture, centralizes enterprise QA knowledge, and enhances predictability of government-grade software deployments.

Failure to implement this solution will result in continued release delays, higher defect rates, increasing labor costs, and difficulty meeting federal traceability requirements across large modernization efforts.

1. Introduction & Background

The enterprise currently conducts QA for government programs through a mixture of manual and semi-automated approaches, using tools such as Selenium, Postman, spreadsheets, and UAT sign-offs. These disparate workflows create long cycle times, inconsistent quality, and fractured auditability.

The AI-Driven QA Automation Platform will automate functional, integration, and regression testing; ingest requirements; auto-generate scripts; self-heal tests when UI or logic changes; and

continuously improve outcomes through ML-driven learning cycles.

2. Problem Statement

Key challenges include:

1. Manual test creation causing inconsistency and duplication across 40+ programs.
2. Regression cycles delaying releases 3–7 days.
3. Brittle automation scripts that break frequently as UIs and APIs evolve.
4. Fragmented test knowledge and poor cross-program reuse.
5. Difficulty achieving government-grade traceability and audit compliance.
6. Pressure to reduce QA operational costs without risking quality.

3. Product Vision Statement

A self-optimizing AI QA platform that autonomously generates, executes, and maintains test scenarios—turning every test cycle into intelligence, every defect into prevention, and every release into a faster, safer deployment.

4. Goals, KPIs & Success Metrics

Primary Goals

- Automate 70–90% of functional and regression test suites.
- Reduce regression cycle durations from days to hours.
- Improve defect detection rates by 20–35%.

- Ensure end-to-end traceability aligned with federal standards.
- Create a reusable enterprise-wide test asset repository.

KPIs

- Cycle time reduction per release
- % tests fully automated
- % decrease in escaped defects
- AI-generated test accuracy
- Time to detect & remediate script breaks
- Audit pass rates & completeness

5. Scope

5.1 In-Scope

Area	Included
Test Types	Functional, integration, regression
Platforms	Web apps, APIs
AI Capabilities	NLP test generation, self-healing, defect learning engine
Integrations	Jira, ADO, Jenkins, GitHub/GitLab, AWS/Azure DevOps
Compliance	Traceability matrix, immutable audit logs
Security	SSO, FedRAMP alignment

5.2 Out-of-Scope (Phase 1)

- Load/performance testing automation
- Legacy mainframe workflows

- Non-FedRAMP environments

6. Key Features & Functional Requirements

5.1 AI-Generated Test Scripts

- NLP ingestion of requirements, epics, UAT criteria
- Auto-generation of functional/integration/regression tests
- Edge case generation via defect pattern mining
- Export to BDD/Gherkin, Selenium, Postman

5.2 Autonomous Test Execution / Orchestration

- Multi-environment orchestration
- Self-healing DOM/API handling
- Parallel execution
- Real-time logs & dashboards

5.3 Continuous Learning Engine

- Flaky test detection
- Defect pattern analysis
- Predictive failure modeling
- Auto-updating enterprise test repository

5.4 Compliance & Traceability Layer

- Requirements → Tests → Results traceability
- Immutable audit logging
- Integration with DevOps ecosystem

5.5 Enterprise Integration Layer

- API-first architecture
- CI/CD integration
- Data masking
- SSO / FedRAMP controls

7. User Personas

1. QA Engineers

Need automated tooling and reduced test maintenance.

2. Product Owners / BAs

Need visibility, traceability, and timely regression summaries.

3. Dev Leads / Architects

Need fast feedback and resilience to API/UI changes.

4. Federal Compliance Officers

Need audit-ready test documentation and reproducible evidence.

8. Use Cases & User Stories

Use Case 1: Automated Test Generation

As a QA Engineer, I want the platform to ingest user stories and auto-generate Selenium/Postman test cases so that I can reduce manual scripting time.

Use Case 2: Regression in Hours

As a Product Owner, I want regression tests executed in less than 4 hours so releases are not delayed.

Use Case 3: Audit Readiness

As a Federal Auditor, I need a traceability matrix showing requirement → test → execution result so I can validate compliance.

Use Case 4: Self-Healing Scripts

As a Dev Lead, I want automation scripts to adjust when UI locators change to prevent regression failures caused by cosmetic UI updates.

Use Case 5: Enterprise Reuse

As a Program Director, I want reusable AI-generated test assets so teams across 40+ programs avoid duplication.

9. Non-Functional Requirements (NFRs)

Security

- FedRAMP High or Moderate-aligned controls
- Encrypted logs, test data, and results
- RBAC and SSO compliance

Performance

- Execute 90% of regression suites within defined SLA windows
- Parallelization scalable to enterprise workloads

Reliability / Availability

- 99.5% uptime target for execution engine
- Auto-retry for failed test orchestrations

Scalability

- Ability to support 40+ concurrent enterprise programs

- Horizontal scaling of execution clusters

Auditing & Logging

- Immutable logs
- Versioning of tests, scripts, and all AI-driven changes

Cost Optimization

- Auto-scaling based on usage
- Cost reporting for execution jobs

10. Assumptions & Dependencies

- Requirements exist in structured formats (Jira/ADO).
- Access to test environments and masked datasets is provided.
- DevSecOps teams support CI/CD integration.
- Governance approves use of AI for QA automation.

11. Risks & Mitigations

- **Weak early AI-generated tests** → Human verification cycles
- **Audit concerns** → Detailed explainability reports
- **Slow user adoption** → Training & hybrid workflows
- **Sensitive data exposure** → Masking, private model hosting

12. Governance & RACI

Role	Responsibility

Product Manager	A
QA Lead	R
DevSecOps	R
Architecture	C
Compliance	C/I
Engineering	R

R = Responsible, A = Accountable, C = Consulted, I = Informed

13. Open Questions & Decisions Needed

1. Will the system host models on-prem, in AWS, or hybrid?
2. Will self-healing changes auto-apply or require approval gates?
3. Which program(s) are pilot candidates for Phase 1?
4. Should the traceability matrix integrate directly with Jira/ADO or generate separate artifacts?

14. Release Plan with Exit Criteria

Phase 1 (0–3 months)

- Requirements ingestion
- Basic NLP test generation
  Exit Criteria: All test cases passed
- 50+ test cases generated from live stories
- Working prototype integrated in at least one DevOps pipeline

Phase 2 (3–6 months)

- Self-healing automation
- Regression suite generation
- CI/CD orchestration
  Exit Criteria: All test cases passed
- Automated regression for at least one application
- Self-healing reducing script fixes by 30%

Phase 3 (6–12 months)

- Enterprise rollout
- Compliance layer
- Continuous Learning engine
  Exit Criteria: All test cases passed
- Traceability reporting validated by compliance
- 20% defect detection improvement in pilot programs