# AI-Driven QA Workflow Upgrade Report

## 1. Competitive Landscape Analysis

The market for AI-driven Quality Assurance (QA) is in a state of rapid evolution, yet it remains highly fragmented. A complete, end-to-end solution that takes a raw User Story and produces both a manual and an automated test plan with full traceability is not currently available as a single, off-the-shelf product. The landscape is instead populated by specialized point solutions that excel in specific stages of the QA lifecycle. A viable strategy for a custom workflow involves integrating best-of-breed open-source tools with commercial platforms via well-defined APIs to create a bespoke, cohesive system.

### 1.1. AI-Powered Test Generation & Planning Platforms

Several vendors have introduced AI capabilities to accelerate test creation and management. testRigor, a prominent platform, is a strong contender for generating the "Automation plan" component of the workflow. The platform claims to enable test creation 100 times faster from plain English instructions and to require 200 times less maintenance.1 It operates by translating high-level, natural language instructions like "purchase a Kindle" into a sequence of executable steps, a process analogous to the proposed "atoms" structure.1 For large enterprises with stringent security and regulatory requirements, testRigor also provides an on-premise solution.2

Another key player is mabl, which focuses on providing AI-driven features for test automation. Its integration strategy is noteworthy, offering a dedicated @mablhq/playwright-tools package. This allows users of Playwright, the open-source automation framework, to incorporate mabl's advanced functionality directly into their test suites. For example, the package supports GenAI Assertions, which enable the validation of complex elements like images, text, and chatbot interactions without the need for intricate, custom code.3 mabl's pricing model is credit-based, but includes unlimited local and Continuous Integration (CI) test runs at no additional cost, which de-risks adoption for teams committed to a "shift-left" testing strategy.4

SpiraTest, a comprehensive Application Lifecycle Management (ALM) tool, has also integrated AI to automate the creation of project artifacts from requirements. This functionality, which leverages Amazon Bedrock and Amazon Nova, can quickly generate sets of standard test cases and Behavior Driven Development (BDD) Gherkin scenarios that can then be refined by a human.5 While this aligns directly with the goal of generating test plans from requirements, it is part of a larger, monolithic suite, which may introduce complexity and vendor lock-in.

The data consistently demonstrates that AI is a powerful accelerant for QA workflows, but it is not a complete replacement for human judgment. An experimental study found that AI-generated test cases provided an impressive 80.07% time efficiency and a 96.11% consistency score in adhering to a structured format.6 However, the same study revealed a moderate ambiguity score of 27.22% on average, highlighting that AI-generated outputs still require human oversight to detect unclear logic or missed scenarios.6 This suggests a causal relationship: the AI handles the rapid, low-value task of initial draft generation, freeing up human QA professionals to focus on high-value activities such as refining prompts, vetting the output for business logic gaps, and improving the model through feedback. This elevates the role of the QA from a test case writer to a strategic architect and reviewer, directly supporting the "human-in-the-loop" requirement of the proposed workflow.

### 1.2. Requirements Vetting & Quality Platforms

The quality of a test plan is fundamentally dependent on the clarity of its source requirements. The research confirms that ambiguities in user stories and requirements documents are a leading cause of project failures.8 The good news is that this is a well-studied problem with existing commercial solutions. QVscribe, for instance, is an automated tool that integrates with established platforms like DOORS Next and Jama to improve the consistency of requirements statements.9 It provides features such as color-coded error detection and unit and term consistency checks.9

A particularly close match to the proposed workflow is ScopeMaster, which uses AI to automatically read, analyze, and even "size" user stories.10 This tool can identify and help correct up to nine categories of potential defects in requirements and can automatically generate test cases that are traceable to the functional intent.10 The existence of such tools provides a clear architectural blueprint to borrow from. Instead of building a complex Natural Language Processing (NLP) pipeline from scratch, the proposed system could emulate or integrate with a "requirements-as-a-service" API. An internal agent could use prompt engineering to instruct a model to "Think like a senior quality analyst" and "Check for boundary conditions and exceptions" to analyze raw user stories.7 This dual-check approach, comparing the model's output to the structured data generated by a dedicated tool, can significantly increase confidence in the requirements and provide a clear gap analysis.

### 1.3. Traceability & Test Management Ecosystems

The value of automated testing is realized only when results are centrally managed and traceable. The research highlights several platforms that serve as a robust reporting layer for automated tests. Xray, for example, is built directly inside Jira and is a first-class citizen of the Atlassian ecosystem.11 It supports both manual and automated tests, and its well-documented REST API provides a standard endpoint (

/rest/raven/1.0/import/execution/junit) for importing test results in JUnit XML format.12 Similarly, TestRail offers a CLI to integrate Playwright test results, and it supports custom properties to embed valuable debugging information, such as screenshots of failures, directly into the report.15

A pattern emerges from this analysis: the test management system acts as a destination for auditable results rather than a central orchestrator. The proposed workflow can be designed with this one-way communication in mind, where the AI agent and Playwright codegen handle the creation and execution, and the CI/CD pipeline pushes the final results to a test management system. This decouples the core AI logic from the reporting layer, simplifying the overall architecture. A notable new entrant is aqua, which offers on-premise deployment and a "live sync" with Jira for requirements and defects, with AI features that create test cases from requirements, prioritize tests, and remove duplicates.16

### 1.4. Visual, UX, and Data-Driven Tools

Visual and user experience (UX) validation represents a specific and challenging aspect of testing. The research indicates that AI-powered solutions are emerging to address this. Applitools, for instance, uses AI to compare screenshots and detect visual differences, a capability that replaces brittle, pixel-based comparisons with an intelligent system that understands UI components.17 Its pricing model is based on "Pages" or "Components" rather than per execution, making it cost-effective for frequent testing.19 The platform also offers on-premise deployment as an optional add-on, which is an important consideration for enterprises with specific data sovereignty needs.19 Percy, another tool, focuses on seamlessly integrating visual testing into CI/CD pipelines, automatically capturing screenshots and highlighting differences using advanced algorithms.17

The concept of an "oracle" in a test "atom" is a core component of the user's vision. These AI-driven visual testing tools provide a sophisticated solution to this problem for UI validation. The proposed workflow can adopt a hybrid Assert model: using native Playwright assertions (expect(locator).toHaveText) for functional and deterministic checks, while relying on external, AI-driven APIs (e.g., Applitools' Visual AI) for visual and non-functional validation. This approach reduces the complexity of writing and maintaining complex assertions in-house.

### 1.5. Comparison Rubric Table

| Vendor/Tool | Category | Key Capability | How close to our backbone (0–5) | Playwright first-class? | Selector strategy guidance? | “Project standards” support? | Pricing (ballpark) | Hosting | Security/PII notes | Integrations | Pros | Cons | Source/Date/Link |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **testRigor** | AI Test Gen | Plain English codegen, AI-driven test creation from US | 4 | No | Yes (via plain English) | Yes (via plain English) | Tiered ($300-900+/mo), Free, Enterprise | SaaS, On-prem | All tests public on free tier 2 | Jira, Zephyr, TestRail 20 | No-code, stable tests, NLP-based 1 | Not Playwright-native; lock-in risk 21 | 1 |
| **mabl** | AI Test Gen | GenAI Assertions, adaptive auto-healing | 3 | Yes (via package) | Yes (auto-healing) | Yes (shared libraries) | Credit-based, Unlimited CI/Local runs 4 | SaaS | Public Cloud Deployment 19 | CI, Jira, Slack 4 | Strong AI, Playwright integration 3 | Vendor lock-in risk, hosted only (by default) | 3 |
| **SpiraTest** | Test Mgmt | US & Test case generation (Gherkin) | 2 | No | No | Yes (ALM templates) | Tiered ($120+/mo) | Cloud, On-prem | On-prem for security/regulatory needs 23 | Jenkins, GitHub, GitLab 5 | All-in-one ALM suite 5 | Monolithic, not Playwright-native 5 | 5 |
| **ScopeMaster** | Req Quality | US review, sizing, defect analysis, test case gen | 4 | No | No | No | Not available, likely custom | SaaS | Not specified | Not specified | High-fidelity US analysis 10 | Focuses only on requirements; not a full workflow tool 10 | 10 |
| **QVscribe** | Req Quality | Requirements quality analysis, consistency checks | 3 | No | No | Yes (customizable configs) | Not available | SaaS | Not specified | MS Office, DOORS, Jama 9 | Integrates with legacy tools 9 | No test generation features 9 | 9 |
| **Xray** | Test Mgmt | Test management within Jira, REST API for results | 5 | Yes (via JUnit) | No | Yes (Jira fields, templates) | Varies (Jira app) | Cloud, Data Center, On-prem | GDPR posture is part of Atlassian cloud 11 | Jira-native, CI/CD, REST APIs 12 | Deep Jira integration 11 | Can be complex to set up 12 | 11 |
| **TestRail** | Test Mgmt | Test case mgmt, rich reporting, CLI for results | 5 | Yes (via CLI/JUnit) | No | Yes (templates, configs) | Varies, Custom | Cloud, On-prem | On-prem option for compliance 15 | Jira, CI/CD, API 11 | Flexible integrations, robust reporting 11 | Can be disconnected from dev workflow 11 | 11 |
| **Applitools** | Visual QA | AI-powered visual testing, UX validation | 3 | Yes (SDKs) | No | No | Tiered ($699-969+/mo), Free | SaaS, On-prem | On-prem option for security 19 | 50+ SDKs, Playwright, CI/CD 19 | AI visual analysis, cost-effective pricing model 19 | Does not provide a full test management solution | 17 |

## 2. Approaches & Best Practices for QA Workflow

The central theme from the research is that an effective AI-driven QA workflow is not fully automated but intelligently augmented. The most effective approach is to position AI as a strategic enabler, with humans-in-the-loop providing context, judgment, and oversight at critical junctures. This principle applies to every stage, from requirements vetting to test execution analysis.

### 2.1. User Story Vetting: From Ambiguity to Actionable Data

The first step in any robust QA workflow is to ensure the requirements are clear and unambiguous. The research confirms that Large Language Models (LLMs) are highly effective at this task, capable of systematically identifying, categorizing, and resolving up to seven different categories of ambiguity, including semantic, syntactic, and functional issues.24 The process requires a high-quality input to produce a high-quality output. Best practices include providing detailed project context and customizing prompts for precision. A powerful approach is to instruct the AI to "Think like a senior quality analyst with 10 years of experience" to focus the analysis on deeper, more critical angles.7

The proposed workflow's requirement for a "confidence/gaps" output is well-supported by this model. The system can be designed to output a measurable ambiguity score, which directly correlates to the need for human intervention.6 For instance, a high ambiguity score could trigger the human-in-the-loop workflow, flagging the user story for a product owner or QA to clarify. This transforms a qualitative metric (ambiguity) into a quantitative, actionable signal for the workflow. The AI's ability to self-assess its output by providing a confidence score turns it into an automated triage agent. Outputs with a low confidence score are routed for immediate human review, while high-confidence outputs can proceed to the next stage of the workflow, maximizing efficiency and minimizing human bottleneck.

### 2.2. Test Planning: The Arrange-Act-Assert (AAA) Pattern

The "atoms" concept in the user's query maps directly to the foundational Arrange-Act-Assert (AAA) pattern, a best practice for structuring test methods. The pattern clearly separates the test logic into three distinct steps: Arrange (setting up the testing objects and prerequisites), Act (performing the action of the test), and Assert (verifying the result).25 The user's workflow can use this pattern as a standard template for the automation plan, with

Setup as the Arrange phase, Action as the Act phase, and Assert/Oracle as the Assert phase. This provides a robust, universally understood framework for test planning.

The analysis of AI and visual testing tools also suggests a "strict vs. lax" model for test feasibility, which can be implemented in the Assert phase. A "strict" test would rely on a deterministic assertion, such as expect(locator).toHaveText.27 In contrast, a "lax" test might accept a non-deterministic or more brittle check, such as a visual assertion using an external tool like Applitools.17 The planner can be programmed to categorize tests into these buckets based on the rigor of their

Assert atom. This categorization provides a nuanced view of the test suite's health and allows for informed decisions regarding quality trade-offs and risk-based testing. A test that relies on a data-testid is inherently more stable and belongs in the "strict" bucket, whereas a test that checks for an empty state with a visual diff may be more prone to flakiness but is still valuable, placing it in the "lax" bucket. The feasibility of a test is not a binary state but a spectrum of reliability and maintenance costs, and this model makes that clear.

### 2.3. Flakiness: Root Causes and Architectural Mitigation

Test flakiness is a major impediment to CI/CD pipelines and undermines trust in the test suite.28 The research identifies common causes, including race conditions, unstable selectors, shared state, and network unpredictability.30 Playwright's native features and industry best practices provide robust architectural mitigations for these issues.

* **Auto-Waiting & Locators:** Playwright's auto-waiting mechanism intelligently waits for elements to be present, visible, and stable before performing an action, which eliminates a primary source of race conditions.27 The best practice for selector stability is to use dedicated test attributes like  
  data-testid or accessibility-based locators like getByRole instead of brittle CSS classes or auto-generated IDs.27 The AI-driven codegen should be programmed to prioritize these stable selectors.
* **Test Isolation & Data:** To prevent "state contamination," each test should be self-contained and avoid relying on shared resources. Playwright's fixture model is a powerful solution for this, encapsulating setup and teardown logic to ensure a clean environment for every test run.27
* **External Dependencies:** Network unpredictability can be mitigated using Playwright's request interception API to mock or stub external services.34 This avoids failures caused by third-party outages or slow responses.

The analysis suggests that test flakiness is an architectural problem, not a simple coding issue. The solution is not merely adding await statements but designing the entire system for stability. The "Project" layer of the proposed workflow, for example, can be designed to enforce the data-testid standard. The AI-driven codegen can then enforce this rule, and the planner can flag any violations. The result is a consistent, reliable test suite where flakiness is architecturally minimized, building trust in the automation.

## 3. Tech Stacks & Architecture Patterns

The proposed workflow requires a robust and scalable architecture to codify quality standards and manage data. The research indicates a clear pattern: use human-readable configuration files (YAML) to define the system's rules and data contracts, and leverage Playwright's native features (fixtures) to build a powerful and maintainable automation layer.

### 3.1. Rules Engine & Planner Implementation

YAML is an ideal choice for the workflow's data contracts, such as US\_Normalized.yaml and Rules YAML. It is a data serialization language designed to be both human-readable and machine-efficient. Its indentation-based syntax makes it easier to read and edit than JSON, particularly for non-developers, and it is the de-facto standard for DevOps configurations like Docker and Kubernetes.36

The Rules YAML can function as a schema that defines the logic for the planner. For example, a rule could be: if US.type == 'Functional' then generate plan for 'Happy Path' and 'Negative Cases'.38 This approach separates the logic of the planning from the code that executes it, making the system more modular and easier to maintain. This separation is crucial for a scalable and adaptable workflow. A key aspect of this architecture is the concept of provenance. The YAML data contract can store metadata, such as the

AI\_model\_id or the human\_reviewer\_id, creating an auditable trail of the plan's creation. This allows the system to trace the origin of any plan that later causes issues, which is essential for regulated industries and high-stakes projects.

### 3.2. Playwright Code Generation & Structure

The quality of the generated code is paramount. The traditional Page Object Model (POM) often leads to code duplication, messy inheritance, and a lack of conformity, as it combines page structure with user interactions.40 A more modern and superior architectural choice is the Screenplay Pattern, which provides a stronger separation of concerns. In this pattern, page classes contain only locators, while user interactions are written as reusable

Task or Question classes that can work with any locator.40

The proposed codegen agent should generate code that follows this Screenplay-like structure. This ensures the generated code is not only functional but also maintainable, scalable, and resistant to flakiness. The value of the AI-driven workflow will be judged not just on its speed but on the quality of its output. Playwright's native fixture model is also a critical component for this architecture. Fixtures are superior to traditional before/after hooks because they encapsulate setup and teardown logic in a single, reusable function, are on-demand, and can be composed to create complex behaviors.32 By leveraging fixtures, the generated tests can be clean, isolated, and easy to maintain.

### 3.3. Configuration & Data Management

The "Project" layer, which defines per-project standards, can be implemented as a set of structured JSON or YAML files.41 YAML is particularly suitable for this because its human-readable syntax and support for comments make it accessible to both technical and non-technical team members.36 This aligns with the best practice of separating test logic from test data, improving maintainability.43

The precedence merge concept is a core principle for this layer's architecture. The system should apply a simple cascade: User Story overrides Project overrides Defaults. This ensures that the most specific, contextual data from the user story takes precedence over general project standards. Any conflicts that arise during the merge can be logged as part of the provenance trail. This provides a clear, auditable decision-making process. The system also requires a versioning strategy for its schemas, which is a concept directly applicable from database management systems where changes are managed via versioning scripts and migration tools.44 This ensures that the data contracts can evolve over time without breaking the workflow.

## 4. Integrations

For a complex, custom workflow, integration is not about adding a logo; it's about establishing a seamless, reliable data flow. The research indicates that the most reliable integrations rely on industry-standard data formats, REST APIs for communication, and a standardized CI/CD environment using containerization.

### 4.1. Test Management & Traceability

The proposed workflow can integrate with popular test management systems like TestRail and Xray by leveraging the industry-standard JUnit XML format. The standard Playwright JUnit reporter can generate a report file with test results.12 Xray provides a dedicated REST API endpoint (

/rest/raven/1.0/import/execution/junit) to import these results into Jira.13 This pattern allows the CI pipeline to push test results without manual intervention, treating the test management tool as a passive reporting layer rather than a central orchestrator.

To establish end-to-end traceability, the test case in Playwright must be linked to a requirement in Jira. The analysis shows that in Azure DevOps, for example, this link is formed when a test case is added to a "Requirement-based test suite".46 A similar approach can be used with Xray or TestRail, where the test report's metadata includes the Jira ticket ID, allowing the system to link the execution status back to the original user story.

### 4.2. CI/CD Pipeline Integration

Playwright integrates seamlessly with major CI providers, including GitHub Actions and Jenkins.47 A standard pipeline for Playwright tests includes steps for checking out the code, installing npm dependencies (

npm ci), and installing the necessary browser binaries (npx playwright install --with-deps).47

For an on-premise or private cloud environment, the same pipeline steps can be executed on self-hosted CI runners.49 The key to ensuring consistency and reproducibility across different environments is containerization. Using official Playwright Docker images ensures that the same browser versions and dependencies are used for both local and CI runs, mitigating issues caused by environmental differences.47 The user's request for on-premise options is not just about technical control; it is also about data sovereignty and compliance. By running the core AI agents and data processing on a self-hosted runner, the system can guarantee that sensitive data, such as PII in user stories, never leaves the private network, which is a critical mitigation for legal and security risks.

### 4.3. Secrets & Security

Secrets management is a crucial component of any automated workflow to handle sensitive information like API keys, database credentials, and user credentials.51 The research emphasizes that secrets should never be hardcoded in files or committed to a Git repository.51 Instead, best practices dictate using a dedicated secrets management system (e.g., HashiCorp Vault, cloud provider secrets managers) and injecting secrets into the CI/CD pipeline at runtime, following the principle of "least privilege".52

A core component of the proposed workflow must be a dedicated PII detection and redaction agent. The research shows that services like Azure AI Language can be used to identify and redact sensitive data, including phone numbers, addresses, and financial information.53 This agent can act as a pre-processing step, sanitizing the user story before it is exposed to any external, hosted AI service. This is a critical, proactive security measure that protects against data leaks and ensures compliance with regulations like GDPR.53

## 5. Risks & Gaps + Mitigations

Building a custom, bespoke workflow introduces inherent risks that must be addressed architecturally from the outset. The key is to design the system with built-in mitigations rather than reacting to failures in production.

### 5.1. Legal, Security & PII Risks

* **Risk:** Sending Personally Identifiable Information (PII) or other sensitive data from user stories to hosted AI services (like OpenAI or Amazon Bedrock) poses a significant legal and security risk, particularly under regulations such as GDPR.53 There is no guarantee that this data will not be used for model training or exposed in a data breach.
* **Mitigation:**
  1. **On-Premise or Private Cloud Hosting:** Run the LLM agent on a self-hosted server or within a private virtual network. This ensures that sensitive data never leaves the organization's control.16
  2. **PII Redaction Agent:** Implement a dedicated pre-processing step using a PII detection service (e.g., Azure AI Language) to automatically remove or mask sensitive information before it touches the AI model.53
  3. **Data Policy:** Establish a clear policy for handling test data, prioritizing synthetic or anonymized data over real production data.53

### 5.2. Vendor Lock-in

* **Risk:** Relying on a single commercial AI-driven platform for test generation creates vendor lock-in. Switching providers can become prohibitively expensive and time-consuming if the vendor uses proprietary technologies or data formats.22
* **Mitigation:**
  1. **Prioritize Open-Source:** Build the core automation backbone (codegen, planner) on open-source tools like Playwright and Node.js. This provides a high degree of control and flexibility.22
  2. **Standard APIs and Formats:** All data contracts and reports should use standard formats like YAML, JSON, and JUnit XML. This ensures that the system components are interchangeable and portable.37
  3. **Abstraction Layer:** Abstract vendor-specific functionality behind internal libraries. The mabl-playwright-tools package is a good example of this, where external functionality is provided through a modular, isolated interface.3

### 5.3. Selector & Environment Instability

* **Risk:** Without a strong architectural foundation, tests will be flaky and unmaintainable. The AI might generate brittle selectors or a plan that fails due to environmental differences.30
* **Mitigation:**
  1. **Project-wide Selector Strategy:** Enforce a team-wide standard of using stable locators like data-testid or getByRole.27 The AI's codegen should be programmed to prioritize these selectors.
  2. **Centralized Configuration:** Manage all environment-specific variables (URLs, secrets) through a centralized Project layer and securely inject them into the pipeline via environment variables or a secrets manager.47
  3. **Containerization:** Use Docker to containerize the entire test environment. This ensures consistency between local and CI runs, making tests more reliable and reducing debugging time.22

## 6. Recommendations & Roadmap

The user's vision for an AI-driven QA workflow is a complex, bespoke system. The most effective approach is a phased roadmap that focuses on quick wins to provide immediate value while simultaneously laying the groundwork for the core architectural components.

### 6.1. Immediate Wins (2-4 Weeks)

* **Recommendation 1: Codify the "Project" Layer.** Create a simple YAML file to store and version project-wide standards for selectors (data-testid), messages (e.g., success toasts, empty states), and regex patterns. This file serves as the first step in building the provenance layer, ensuring a single source of truth for project-specific rules. If YAML is not preferred, a simple JSON file can be used as an alternative, though it is less human-readable.
* **Recommendation 2: Implement the Arrange-Act-Assert Pattern.** Modify the Plan CSV output to be structured into explicit Setup, Action, and Assert columns. This provides a clear, auditable structure and enforces the fundamental AAA pattern from the beginning.25 As an alternative, a YAML-based output could be used to support more complex, nested data for actions or assertions, which would be more scalable in the long term.

### 6.2. Strategic Roadmap (3-6 Months)

* **Phase 1: Build the Core Agents (Months 1-2).** Prototype the US Review agent using an internal LLM wrapper. This core intellectual property should not be outsourced. The prompt engineering and PII handling are unique to the organization's data. As part of this phase, integrate a PII redaction service as a pre-processing step to sanitize all user stories before they are sent to the AI model.53
* **Phase 2: Integrate a Planning Rules Engine (Months 2-4).** Develop a simple Node.js/TypeScript script that reads the US\_Normalized.yaml and a Rules.yaml file. The script's logic should apply business rules to generate the Plan CSV, which can include a "strict vs. lax" bucket selection for tests.25 This core logic should also be built in-house to maintain full control.
* **Phase 3: Architect the Codegen Layer (Months 4-6).** Develop a Playwright codegen agent that takes the Plan CSV as input and produces Playwright tests. The generated code must follow the Screenplay pattern to ensure it is maintainable and scalable. It should use the Project layer for standardized locators and messages. This is the final output of the system and must be built to exact specifications to avoid the maintenance burden of a black-box vendor solution.

### 6.3. Data Contract Evolution

* **US\_Normalized.yaml:** This file should be evolved to include a confidence\_score (from 0 to 100) and ambiguity\_notes as outputs from the LLM agent.6 It should also contain a  
  provenance block with fields such as model\_version, prompt\_version, and timestamp to ensure a full audit trail.
* **Rules.yaml:** This file will codify the if/then logic for plan generation and the selection criteria for "strict vs. lax" tests.
* **Plan.csv:** The structure should be updated to include columns for Atom\_ID, Step, Type (Setup, Action, Assert), Locator, Data, and Feasibility (Strict, Lax).

### 6.4. Proof-of-Concept Ideas

* **PoC #1: The "Project" Layer.** The goal is to validate the architecture of the shared standards layer. This can be accomplished by creating a Project-standards.yaml file with a few standardized selectors (data-testid) and messages. A simple Playwright test can then be written to use a helper function that reads from this file, demonstrating that the Project layer can be a single source of truth for the entire team.
* **PoC #2: LLM-based Confidence Scoring.** The goal is to validate the AI's ability to self-assess its output. A set of 20 user stories (10 ambiguous, 10 clear) can be submitted to a model with a prompt that asks it to both normalize the story and provide a confidence score (0-100) on its output. The AI's scores can then be compared to a human's assessment to validate the reliability of this output.

#### Works cited

1. AI-Based Test Automation Tool [2025] - testRigor Software Testing, accessed September 3, 2025, <https://testrigor.com/>
2. testRigor Reviews [2025] – Pros, Cons, Features and Pricing - Test Automation Tools, accessed September 3, 2025, <https://testautomationtools.dev/testrigor-reviews/>
3. Enhancing Playwright tests with mabl – mabl help, accessed September 3, 2025, <https://help.mabl.com/hc/en-us/articles/31272435532692-Enhancing-Playwright-tests-with-mabl>
4. mabl Pricing, accessed September 3, 2025, <https://www.mabl.com/pricing>
5. Features of SpiraTest, Our Test Management Software. Try It Today!, accessed September 3, 2025, <https://www.inflectra.com/Products/SpiraTest/Highlights.aspx>
6. AI-generated test cases from user stories: An experimental research study - Thoughtworks, accessed September 3, 2025, <https://www.thoughtworks.com/en-us/insights/blog/generative-ai/AI-generated-test-cases-from-user-stories-an-experimental-research-study>
7. Can we use generative AI to generate test cases from user stories? - Thoughtworks, accessed September 3, 2025, <https://www.thoughtworks.com/en-us/insights/blog/generative-ai/can-we-use-generative-AI-to-generate-test-cases-from-user-stories>
8. Avoid Ambiguity in Your User Story - Business Analysis Experts, accessed September 3, 2025, <https://www.businessanalysisexperts.com/user-stories-guide-writing-requirements/avoid-ambiguity-user-story/>
9. QVscribe - Features & Pricing (August 2025) - SaaSworthy, accessed September 3, 2025, <https://www.saasworthy.com/product/qracorp-qvscribe>
10. Best Requirements Management Software for Cloud of 2025 ..., accessed September 3, 2025, <https://sourceforge.net/software/requirements-management/saas/>
11. What is Test Management? A Complete Guide for Modern QA Teams - TestPod, accessed September 3, 2025, <https://testpod.io/what-is-test-management-a-complete-guide-for-modern-qa-teams/>
12. Testing web applications using Playwright - Xray Server + DC ..., accessed September 3, 2025, <https://docs.getxray.app/display/XRAY/Testing+web+applications+using+Playwright>
13. Import Execution Results - REST - Xray Server + DC, accessed September 3, 2025, <https://docs.getxray.app/space/XRAY/301667997/Import+Execution+Results+-+REST>
14. Import Execution Results - REST v2 - Xray Cloud Documentation, accessed September 3, 2025, <https://docs.getxray.app/display/XRAYCLOUD/Import+Execution+Results+-+REST+v2>
15. Integrating with Playwright – TestRail Support Center, accessed September 3, 2025, <https://support.testrail.com/hc/en-us/articles/9682231778324-Integrating-with-Playwright>
16. aqua cloud: AI-Powered Test Management Software, accessed September 3, 2025, <https://aqua-cloud.io/>
17. Top 10 Visual Regression Testing Tools: Ensuring Pixel-Perfect User Experiences - Apidog, accessed September 3, 2025, <https://apidog.com/blog/best-visual-regression-testing-tools/>
18. Visual Regression Testing: Detect UI Issues Before Users Notice | Keploy Blog, accessed September 3, 2025, <https://keploy.io/blog/community/visual-regression-testing>
19. Platform - Pricing - AI-Powered End-to-End Testing | Applitools, accessed September 3, 2025, <https://applitools.com/platform-pricing/>
20. Release Notes: New Integrations, Import Labels, Handle Popups - testRigor AI-Based Automated Testing Tool, accessed September 3, 2025, <https://testrigor.com/releases/release-notes-new-integrations-import-labels-handle-popups/>
21. Playwright vs. Cypress - testRigor AI-Based Automated Testing Tool, accessed September 3, 2025, <https://testrigor.com/blog/playwright-vs-cypress/>
22. What is Vendor Lock-In? 5 Strategies & Tools To Avoid It - Superblocks, accessed September 3, 2025, <https://www.superblocks.com/blog/vendor-lock>
23. Pricing for SpiraTest - Inflectra Corporation, accessed September 3, 2025, <https://www.inflectra.com/Purchase/SpiraTest.aspx>
24. Leveraging Large Language Model for Ambiguity Detection and Resolution in Software Requirements - ProQuest, accessed September 3, 2025, <https://search.proquest.com/openview/d84fd5d649bb03e472ded1cf605851a9/1?pq-origsite=gscholar&cbl=18750&diss=y>
25. Arrange Act Assert | JustMock Documentation, accessed September 3, 2025, <https://docs.telerik.com/devtools/justmock/basic-usage/arrange-act-assert>
26. The Arrange, Act, and Assert (AAA) Pattern in Unit Test Automation - Semaphore CI, accessed September 3, 2025, <https://semaphore.io/blog/aaa-pattern-test-automation>
27. Say Goodbye to Flaky Tests: Playwright Best Practices Every Test Automation Engineer Must Know | by Sam Sperling | Medium, accessed September 3, 2025, <https://medium.com/@samuel.sperling/say-goodbye-to-flaky-tests-playwright-best-practices-every-test-automation-engineer-must-know-9dfeb9bb5017>
28. What is a Flaky Test? Causes, Identification & Remediation - Datadog, accessed September 3, 2025, <https://www.datadoghq.com/knowledge-center/flaky-tests/>
29. How To Manage Flaky Tests in your CI Workflows - The Mill Build Tool, accessed September 3, 2025, <https://mill-build.org/blog/4-flaky-tests.html>
30. Avoiding Flaky Tests in Playwright | Better Stack Community, accessed September 3, 2025, <https://betterstack.com/community/guides/testing/avoid-flaky-playwright-tests/>
31. 9 Playwright Best Practices: Reliable and Easy Tests - Oxylabs, accessed September 3, 2025, <https://oxylabs.io/blog/playwright-best-practices>
32. What is the use of fixtures in Playwright? - testomat.io, accessed September 3, 2025, <https://testomat.io/blog/what-is-the-use-of-fixtures-in-playwright/>
33. Fixtures | Playwright, accessed September 3, 2025, <https://playwright.dev/docs/test-fixtures>
34. How to Intercept Requests in Playwright | Checkly, accessed September 3, 2025, <https://www.checklyhq.com/learn/playwright/intercept-requests/>
35. Network | Playwright Python, accessed September 3, 2025, <https://playwright.dev/python/docs/network>
36. YAML vs JSON - Difference Between Data Serialization Formats - AWS, accessed September 3, 2025, <https://aws.amazon.com/compare/the-difference-between-yaml-and-json/>
37. YAML for Scalable and Simple Test Automation - Codoid, accessed September 3, 2025, <https://codoid.com/automation-testing/yaml-for-scalable-and-simple-test-automation/>
38. YAML schema reference for Azure Pipelines - Microsoft Learn, accessed September 3, 2025, <https://learn.microsoft.com/en-us/azure/devops/pipelines/yaml-schema/?view=azure-pipelines>
39. Add data tests to your DAG | dbt Developer Hub - dbt Docs, accessed September 3, 2025, <https://docs.getdbt.com/docs/build/data-tests>
40. Screenplay vs. Page Objects - Boa Constrictor - GitHub Pages, accessed September 3, 2025, <https://q2ebanking.github.io/boa-constrictor/getting-started/page-objects/>
41. Automation Testing Framework with Playwright and YAML - GitHub, accessed September 3, 2025, <https://github.com/modalqa/playwright-with-yaml>
42. YAML Tutorial : A Complete Language Guide with Examples - Spacelift, accessed September 3, 2025, <https://spacelift.io/blog/yaml>
43. The Complete Guide to Reading different Files in Test Automation | by JigNect - Medium, accessed September 3, 2025, <https://medium.com/@jignect/the-complete-guide-to-reading-different-files-in-test-automation-bea38849fa3a>
44. What is Schema Versioning in DBMS? - GeeksforGeeks, accessed September 3, 2025, <https://www.geeksforgeeks.org/dbms/what-is-schema-versioning-in-dbms/>
45. Reporters | Playwright, accessed September 3, 2025, <https://playwright.dev/docs/test-reporters>
46. Create test plans and suites - Azure Test Plans | Microsoft Learn, accessed September 3, 2025, <https://learn.microsoft.com/en-us/azure/devops/test/create-a-test-plan?view=azure-devops>
47. Integrating Playwright with CI/CD: Jenkins vs GitHub Actions - BrowserCat, accessed September 3, 2025, <https://www.browsercat.com/post/integrating-playwright-with-cicd-pipelines>
48. Implementing End-to-End Testing Using Playwright within Jenkins CI/CD Pipelines, accessed September 3, 2025, <https://blogs.perficient.com/2025/07/15/implementing-end-to-end-testing-using-playwright-within-jenkins-ci-cd-pipelines/>
49. Running Playwright tests in continuous integration - Microsoft Learn, accessed September 3, 2025, <https://learn.microsoft.com/en-us/shows/getting-started-with-end-to-end-testing-with-playwright/running-playwright-tests-in-continuous-integration>
50. Scaling Playwright in CI/CD Pipelines - Testkube, accessed September 3, 2025, <https://testkube.io/resources/scaling-playwright-in-ci-cd-pipelines-playwright-user-event>
51. What Is Secrets Management? - Palo Alto Networks, accessed September 3, 2025, <https://www.paloaltonetworks.com/cyberpedia/secrets-management>
52. Secrets Management - OWASP Cheat Sheet Series, accessed September 3, 2025, <https://cheatsheetseries.owasp.org/cheatsheets/Secrets_Management_Cheat_Sheet.html>
53. What is the Personally Identifying Information (PII) detection feature ..., accessed September 3, 2025, <https://learn.microsoft.com/en-us/azure/ai-services/language-service/personally-identifiable-information/overview>
54. Defining and detecting personally identifiable information (PII) in log data - Splunk Lantern, accessed September 3, 2025, <https://lantern.splunk.com/Security/UCE/Foundational_Visibility/Compliance/Defining_and_detecting_personally_identifiable_information_(PII)_in_log_data>
55. What Is Vendor Lock In and How to avoid It - DashDevs, accessed September 3, 2025, <https://dashdevs.com/blog/how-to-avoid-vendor-lock-in-traps/>