Test Automation Framework: Agent Architecture Description

Alberto Espinosa
KSqure Group

July 21, 2025

1 Scope

This document delineates the agent-based architecture for an automated test generation framework, designed to streamline software testing by converting user stories into executable test cases. The minimum viable product (MVP) focuses on foundational functionality: generating synthetic user stories, parsing inputs, producing Gherkin scenarios, creating and executing test scripts, and reporting code coverage. Advanced features, such as self-healing tests, CI/CD pipeline integration, automated pull request commenting, agentic pull request reviews, and Slack/Teams notifications, are reserved for future iterations. The project is executed by a team of three AI developers, each responsible for two agents. Given the limited team size, additional resources may be required to meet the timeline, with potential replanning if support is unavailable.

2 Agent Architecture Overview

The test automation framework employs a sequential pipeline of six agents, implemented within a LangGraph environment using pre-trained Ollama models. The pipeline transforms project requirements into executable test cases, addressing challenges such as manual testing overhead, inconsistent user story translation, and flaky test detection. The agents operate in sequence: the Automatic User Histories Agent generates synthetic user stories; the Reader Agent parses inputs; the Converter to Gherkin Agent produces Gherkin scenarios; the Generator of Intelligent Tests Agent creates executable test scripts; the Writer Agent organizes test files; and the Runner Agent executes tests and generates coverage reports. The architecture supports use cases such as login, payment processing, and user registration, ensuring scalability and modularity for future enhancements.

3 Agent Descriptions

3.1 Automatic User Histories Agent

Goal: Generate synthetic user stories from scratch to facilitate MVP development, simulating technical requirements for testing scenarios until real inputs are available.

Inputs: Python codebase (text files) or project requirements (JSON/text).

Outputs: Synthetic user stories in JSON format, e.g., {`"story": "As a user, I want to log in with valid credentials to access my account"`}. These outputs are preliminary and subject to change.

Ollama Configuration: DeepSeek model, temperature 0.7 for creative story generation, top-p 0.9.

3.2 Reader (Parser) Agent

Goal: Parse user stories and codebase to extract structured data for test generation, ensuring compatibility with downstream agents.

Inputs: Synthetic user stories (JSON/text) and Python codebase (text files).

Outputs: Structured JSON, e.g., {`"functions": [{"name": "login", "params": ["username", "password"]}], "requirements": ["valid login"]`}. Preliminary format.

Ollama Configuration: DeepSeek model, temperature 0.3 for deterministic parsing, top-p 0.95.

3.3 Converter to Gherkin Agent

Goal: Transform parsed user stories into Gherkin scenarios for use in behaviour-driven development frameworks like Cucumber.

Inputs: Structured JSON from the Reader Agent.

Outputs: Gherkin `.feature` files, e.g.,

```
Feature: Login
Scenario: Valid login
Given a user with valid credentials
When they log in
Then they access their account
```

Playwright/Cucumber format is assumed, with flexibility for alternatives (e.g., Selenium) if desired.

Ollama Configuration: Qwen model, temperature 0.5 for balanced creativity, top-p 0.9.

3.4 Generator of Intelligent Tests Agent

Goal: Generate executable test scripts based on Gherkin scenarios, leveraging AI to ensure comprehensive test coverage.

Inputs: Gherkin `.feature` files from the Converter Agent.

Outputs: Test scripts in Playwright/Cucumber format (e.g., `test_login.js`:

```
const { test } = require('@playwright/test');
test('Valid login', async () => { ... });
```

Format is flexible (e.g., Selenium) if required.

Ollama Configuration: DeepSeek model, temperature 0.4 for precise code generation, top-p 0.95.

3.5 Writer Agent

Goal: Organize generated test scripts and Gherkin files into appropriate project directories for execution.

Inputs: Gherkin `.feature` files and test scripts from the Generator Agent.

Outputs: Organized test files in directories, e.g., `/tests/login.feature`, `/tests/test_login.js`. Directory structure is preliminary.

Ollama Configuration: Qwen model, temperature 0.3 for deterministic file organization, top-p 0.95.

3.6 Runner Agent

Goal: Execute test scripts and generate coverage and test result reports, identifying pass/fail outcomes and untested code.

Inputs: Test files from the Writer Agent.

Outputs: JSON test results and coverage reports, e.g., {`"pass": 90, "fail": 10, "coverage": "85%"`}, and visualization plots (e.g., `coverage.png`). Format is preliminary.

Ollama Configuration: DeepSeek model, temperature 0.3 for consistent reporting, top-p 0.95.

4 Team Composition

Name	Role	Agent	Comments
TeamMember1	AI Developer (Agents)	Automatic User Histories, Reader	Responsible for initial input generation and parsing
TeamMember2	AI Developer (Agents)	Converter to Gherkin, Generator of Intelligent Tests	Manages test case creation
TeamMember3	AI Developer (Agents)	Writer, Runner	Handles test organization and execution

Table 1: Team Members and Assigned Agents