Test Automation Framework: Development Roadmap

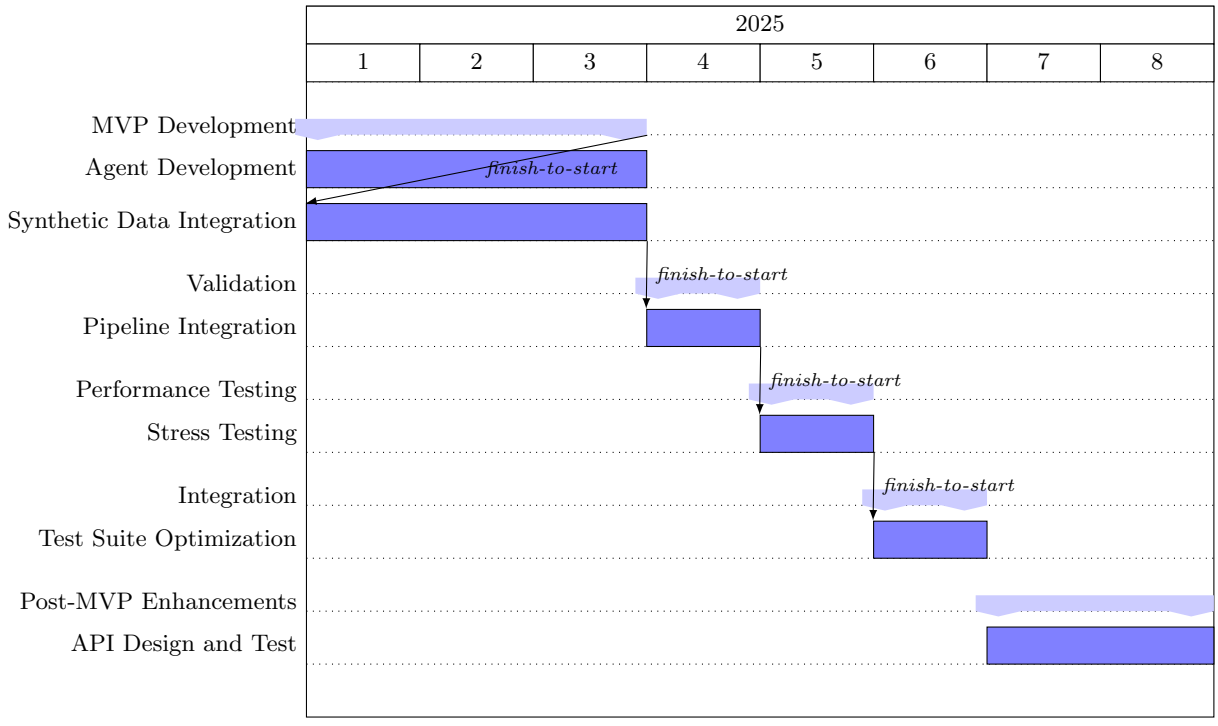
July 21, 2025

1 Scope

This roadmap outlines the development of an agent-based test automation framework, focusing on a minimum viable product (MVP) that generates synthetic user stories, converts them to Gherkin scenarios, creates and executes test scripts, and reports code coverage. The MVP targets basic functionality to support use cases such as login, payment processing, and user registration. Advanced features, including self-healing tests, CI/CD integration, automated pull request commenting, agentic pull request reviews, and Slack/Teams notifications, are designated 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 Roadmap

The roadmap spans eight weeks, from Saturday, 21 July 2025, to Friday, 12 September 2025, organized into weekly blocks.



3 Roadmap Stage Descriptions

This section elucidates each stage of the roadmap, detailing objectives and expected outputs. The stages guide developers through the sequential development of the agent-based test automation framework, ensuring progression towards a functional MVP within eight weeks. Each stage builds upon the previous, culminating in a system that automates test generation and execution for software testing.

3.1 MVP Development (Weeks 1–3)

Objective: Implement the six agents (Automatic User Histories, Reader, Converter to Gherkin, Generator of Intelligent Tests, Writer, Runner) independently, ensuring each functions correctly in isolation outside the LangGraph environment. This includes generating synthetic user stories to simulate technical requirements.

Tasks:

- **Agent Development**: Develop standalone logic for each agent using Python and pre-trained Ollama models, focusing on individual functionality for user story generation, parsing, test creation, and execution.
- **Synthetic Data Integration**: Implement the Automatic User Histories Agent to generate synthetic user stories, ensuring compatibility with downstream agents.

Expected Outputs:

- Functional agents, each producing:
  - JSON user stories (e.g., {`"story": "As a user, I want to log in"`}).
  - Parsed JSON data (e.g., {`"functions": [...], "requirements": [...]`}).
  - Gherkin `.feature` files (e.g.,

    ```
    Feature: Login
    Scenario: Valid login
    ```

).
  - Test scripts (e.g., `test_login.js`: `test('Valid login', async () => {...});`).
  - Organized test files in directories (e.g., `/tests/login.feature`).
  - JSON test results and coverage reports (e.g., {`"pass": 90, "coverage": "85%"`}).

3.2 Validation (Week 4)

Objective: Integrate all agents into a cohesive LangGraph pipeline, ensuring seamless input-output synchronization across use cases such as login and payment processing.

Tasks:

- **Pipeline Integration**: Connect the six agents in a LangGraph environment, validating data flow from user story generation to test execution and reporting.

Expected Outputs:

- A fully integrated pipeline producing a sequence of outputs: JSON user stories, parsed data, Gherkin files, test scripts, organized files, and coverage reports, validated for use cases. Formats (e.g., Playwright, Selenium) are flexible if required.

3.3 Performance Testing (Week 5)

Objective: Evaluate the pipeline's performance and robustness through stress testing across use cases, ensuring reliability under varying conditions.

Tasks:

- **Stress Testing**: Execute the pipeline on use cases (e.g., login, payment processing), assessing test execution stability and coverage completeness.

Expected Outputs:

- Validated pipeline outputs, including test scripts, JSON test results (e.g., {`"pass": 90, "fail": 10`}), coverage reports (e.g., {`"coverage": "85%"`}), and visualization plots (e.g., `coverage.png`).

3.4 Integration (Week 6)

Objective: Optimize the test suite by applying prioritization rules to enhance efficiency and coverage for critical codebase functions.

Tasks:

- **Test Suite Optimization**: Deploy the Runner Agent to apply prioritization rules (e.g., flagging tests for high-risk functions like payment processing), producing an optimized test suite.

Expected Outputs:

- Optimized test suites with prioritization metadata (e.g., {`"test": "login", "priority": "high"`}) and updated test files in directories.

3.5 Post-MVP Enhancements (Weeks 7–8)

Objective: Finalize the system by developing a RESTful API to deliver test results and coverage reports, enhancing accessibility.

Tasks:

- **API Design and Test**: Implement the Writer and Runner Agents to produce unified reports and validate a RESTful API for delivering outputs.

Expected Outputs:

- Unified JSON reports (e.g., {`"tests": [...], "coverage": "85%", "summary": "Robust test suite"`}) and a validated RESTful API for report delivery.

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