The goal of writing unit tests for your software code is to ensure it has been tested against all cases before production. Writing these tests manually can be a time-intensive process. We built an agent that takes an input directory with relevant code files (Python code in this case) and generates unit tests to verify if your code passes all test cases. Below, we'll walk you through how we built this agent.

Inputs

1. Software architecture documents
2. Interface control documents
3. Sample requirements and test cases combinations.

The test generation process includes the following main steps:

- **Data preparation:** Input documents are indexed and stored in an embedding database, which is later used to query relevant information.
- **Requirements extraction:** Requirement details are retrieved from the requirement storage system (for example, Jira).
- **Data traceability:** AUTG searches the embedding database to trace information related to the input requirements. The output is a mapped connection between the requirement and the relevant fragments.
- **Test specification generation:** Based on the verification steps from requirements and the identified fragments (traceability), AUTG generates both positive and negative test specifications to cover all aspects of the requirement.
- **Test implementation generation**: AUTG uses the ICD fragments (traceability) and the generated test specifications to create the tests in Python. The ICD provides context such as function names, data types, enumerations, and return codes, which the LLM uses during code generation. This step results in executable tests.
- **Test execution:** The generated tests are compiled and executed and coverage data is collected. The AUTG agent analyzes test coverage results and repeats the generation of test specifications and implementation for the missing cases.

Given the input requirements and driver documentation, AUTG extracts requirement information from Jama, traces it to the corresponding documentation fragments, and generates test specifications and implementations.

Key Phases

- Requirement extraction
- SWAD data traceability
- ICD data traceability
- Test specification generation
- Test implementation generation

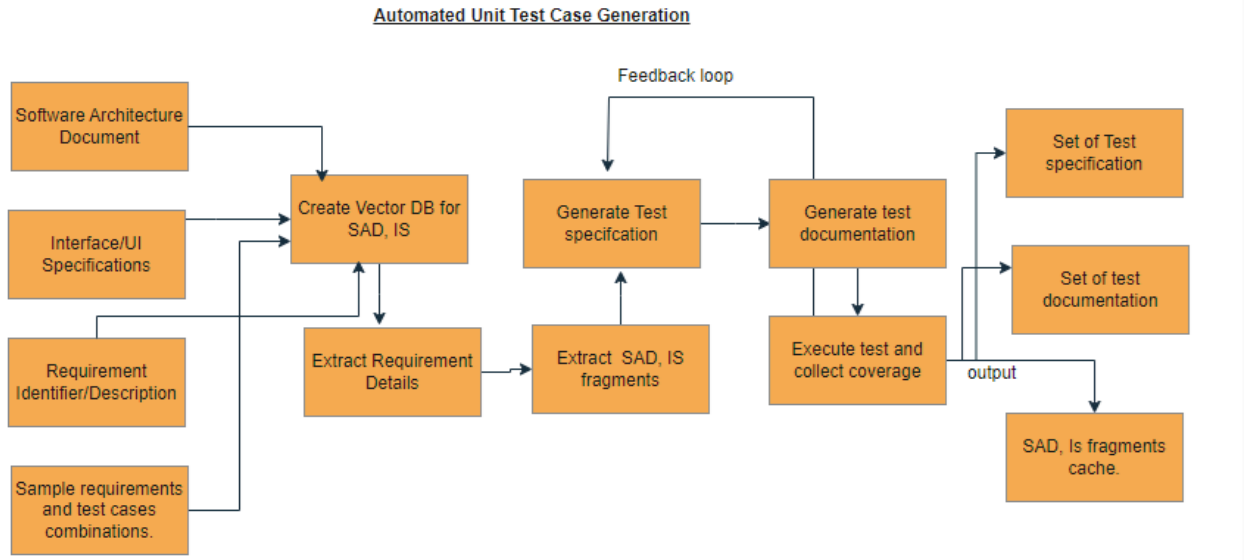
1. Backend Services

- **JIRA API Integration**: Fetches user stories from JIRA when created.
- **CrewAI Orchestration**: Manages task execution between test case generation and documentation agents.
- **LLM API (Qwen or OpenAI GPT-4)**: Generates unit test cases and README files based on user stories.
- **Slack/Webhook Notification Service**: Sends workflow status updates to users.
- **Database (PostgreSQL/Firestore)**: Stores generated test cases and documentation for version control.

2. Data Sources to be Integrated

- **JIRA API**: Fetch user stories.
- **Code Repository (GitHub/GitLab)**: Store test cases and README files.
- **LLM API (Qwen/GPT-4)**: For generating test cases and documentation.
- **Slack API**: Send real-time updates.

Workflow



Code

https://colab.research.google.com/drive/1FuT_1ztl2T1jpTQ23VJ_C2-6pJixumaK?authuser=4

Webhook Integration for Automation

JIRA Webhook Configuration:

- Go to **JIRA Admin > System > Webhooks**
- Set the webhook URL: http://your-server-url/trigger_workflow
- Select event: **"Issue Created"**

This ensures the workflow is automatically triggered when a new **JIRA user story** is created.