**Test Scenario Generator Script Analysis**

**Overview**

The Test Scenario Generator script is an automated tool designed to analyze code modules and generate comprehensive test scenarios. It bridges the gap between implementation and testing by examining code alongside original specifications to create relevant, detailed test scenarios that cover functional behavior, edge cases, and error conditions. This automation ensures consistent test coverage and accelerates the software testing process.

The script works in two key phases:

1. **Code Description Generation**: Analyzes implemented Python modules to extract detailed descriptions of functionality, components, and architecture
2. **Test Scenario Generation**: Combines code analysis with original technical specifications and user stories to create comprehensive test scenarios

By automating the creation of test scenarios directly from implemented code, the tool ensures alignment between development artifacts and testing efforts, reducing manual effort while improving test quality and coverage.

**Input/Output**

**Input**

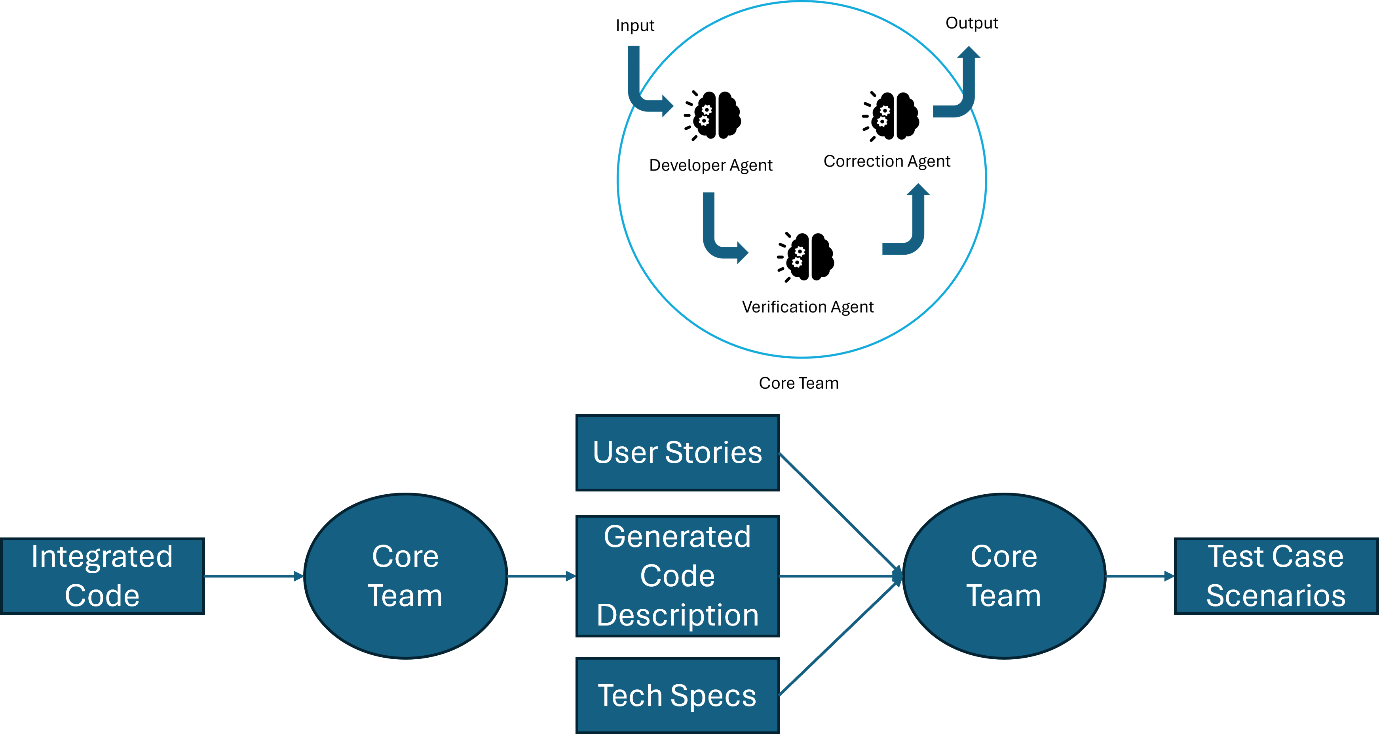
* Path to the latest integrated solution folder containing Python modules (automatically detected)
* Excel file with original technical specifications and user stories
* Environment variables for Azure OpenAI configuration:
  + AZURE\_OPENAI\_API\_KEY
  + AZURE\_OPENAI\_ENDPOINT
  + AZURE\_OPENAI\_API\_VERSION
  + AZURE\_OPENAI\_CHAT\_DEPLOYMENT\_NAME

**Output**

* A timestamped directory (test\_scenarios\_YYYYMMDD\_HHMMSS/) containing:
  + **Code Descriptions**: JSON files with detailed analysis of each module's functionality, architecture, and components
  + **Test Scenarios**:
    - JSON files with structured test scenarios for each module
    - Markdown files with formatted, human-readable test scenarios
  + **Summary Report**:
    - JSON format for programmatic access
    - Markdown format with tables summarizing test coverage across modules

**Process Flow**

**Process Flow Design**



**Execution Flow**

1. **Initialization and Configuration**
   * main() - Entry point function
   * check\_openai\_config() - Verify Azure OpenAI credentials
2. **Solution Discovery and Analysis**
   * find\_integrated\_solution\_folder() - Locate integrated solution
   * find\_code\_files() - Identify all module files (.py)
   * create\_output\_directory() - Create timestamped directory structure
3. **Code Description Generation**
   * process\_code\_files() - For each code module:
     + CodeDescriptionGenerator uses AI to analyze the code:
       - analyzer() - Extract key components, architecture, and functionality
       - \_extract\_json\_from\_response() - Parse structured description
       - \_save\_description() - Store as JSON
4. **Information Gathering**
   * read\_tech\_specs\_and\_user\_stories() - Extract from Excel:
     + Original requirements
     + User stories with business context
     + Map to appropriate module IDs
5. **Test Scenario Generation**
   * generate\_test\_scenarios() - For each module:
     + TestScenarioGenerator combines code descriptions with requirements:
       - generator() - Create scenarios covering functional areas and edge cases
       - \_extract\_json\_from\_response() - Parse structured scenarios
       - \_save\_test\_scenarios() - Store as JSON and Markdown
       - \_convert\_to\_markdown() - Format for human readability
6. **Reporting**
   * create\_summary\_report() - Generate overview of:
     + Total modules analyzed
     + Test scenarios created
     + Coverage metrics
     + Save in both JSON and Markdown formats

**Input Fields Description**

|  |  |  |  |
| --- | --- | --- | --- |
| **Field Name** | **Description** | **Input Type** | **Validation** |
| excel\_file\_path | Path to Excel file with technical specifications and user stories | String | File must exist and be a valid Excel format |
| solution\_folder | Path to integrated solution folder (auto-detected) | String | Directory must exist and contain module files |
| code\_files | List of paths to US\_\*\_code.py files | List[String] | Files must exist and be valid Python files |
| AZURE\_OPENAI\_API\_KEY | API key for Azure OpenAI service | String | Must be non-empty |
| AZURE\_OPENAI\_ENDPOINT | Endpoint URL for Azure OpenAI service | String | Must be a valid URL |
| AZURE\_OPENAI\_API\_VERSION | API version for Azure OpenAI service | String | Must be a valid version string |
| AZURE\_OPENAI\_CHAT\_DEPLOYMENT\_NAME | Deployment name for Azure OpenAI chat model | String | Must be a valid deployment name |

**Issues/Error Handling**

|  |  |  |
| --- | --- | --- |
| **Sr No** | **Issue/Error** | **Solution** |
| 1 | Missing Azure OpenAI configuration | Script attempts to set default values; raises detailed EnvironmentError if configuration is still missing |
| 2 | No integrated solution folder found | Raises FileNotFoundError with clear message about missing integrated solution folder |
| 3 | No code files found in solution folder | Logs error message and exits gracefully with informative return value |
| 4 | Failed to extract user story ID from filename | Logs warning and skips processing that file; continues with others |
| 5 | Failed to extract JSON from AI response | Implements fallback structure with error details; continues processing |
| 6 | Error during code analysis | Catches exception, logs detailed error, continues with other modules |
| A | Missing or invalid Excel file | Provides informative error about Excel file issue with path details |
| 8 | API rate limiting from Azure OpenAI | Built-in retry mechanism with exponential backoff; logs retries |
| 9 | Failed to generate test scenarios | Logs specific error for the problematic module; continues with others |
| 10 | Unable to extract user story ID from Excel | Attempts multiple regex patterns; logs warning if all fail |

The script implements comprehensive logging throughout with detailed error messages and graceful degradation, ensuring that failures in processing individual modules don't prevent the overall workflow from completing. The error handling approach prioritizes completing as much of the process as possible while providing clear diagnostics about any issues encountered.