Additional Documentation – Test Plan, Testing strategy, workflow diagram

Revision History

Revision #	Date	Author	Description
1.00	2016-Aug-04	Ridhwaan Shakeel	Initial Creation

Test Plan Approvals

Name	Role	Approval Date	Responsibility
Ridhwaan Shakeel			Creator

Purpose

This document contains a basic test plan, workflow diagram, strategies, some test scenarios used to test the compare_forecasts.py program for comparing 5 day weather forecasts between two cities

Intended Audience

This document is intended for the SQA team, reviewers, and those to whom it may concern

Scope

The scope of testing will consist of the compare_forecasts.py Python program and the openweather.org/forecast5 weather API integration testing. The tests shall consist of user input on the terminal CLI of compare_forecasts program, as well as requests sent and responses received from the weather forecast API. User request of city name and forecast time from Python forecast program shall propagate to the weather forecast API and likewise weather 5 day forecast responses from API shall propagate to the Python program. For testing purposes, the terminal tests are run manually; workflow can be automated to use cronjob/ bash script.

Assumptions

This test scenario document assumes that the compare_forecasts program and weather forecast API provides a basic module interface and error reporting capabilities. Also assume that all Python module user actions occur from the terminal and the weather API responses are always in JSON with UTC forecast times.

Test plan, Testing strategy

The program I worked on has functionality and validation, so for testing, I plan to limit the scope to:
- verification and functional black box testing, to check if the program met the requirements of the assignment,
- integration testing- test the openweather forecast5 API with the Python module
- As a system tester, I can assume there will be user acceptance testing by the intended audience

Rule of thumb I will follow is to test zero case, positive case, negative case (i.e. non-existent city name, malformed/ bad IO), and crud test. But because this particular Python program doesn't have full CRUD, I will need to test only the Read / GET weather forecast portion. Also test if the temperature format is correct and weather description came from the response content.

The test cases for this assignment need to test good and bad input and output for coverage. I have to translate use cases into unit tests that are simple, isolated and don't overlap. With the error reporting and validation logic in the code, it will be easy to see the actual results or stack trace for success and failure criteria.

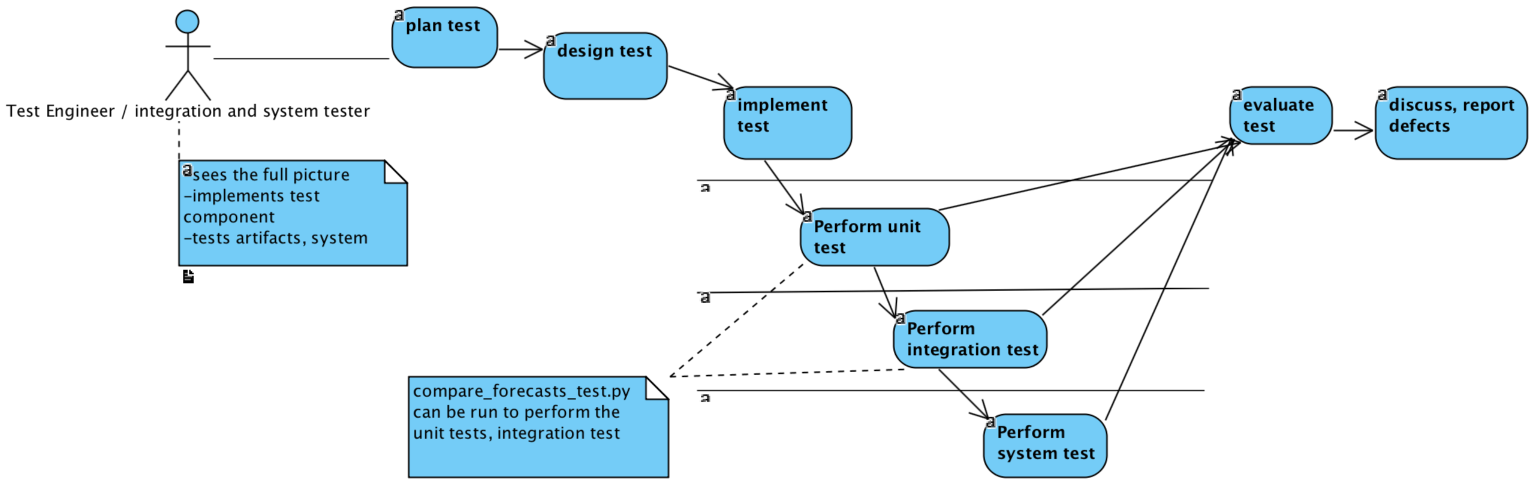
As the Python weather forecast program uses the Requests library against a real API service, I need to mock the http calls in order to unit test bad requests or simulate bad outputs (i.e. how will the program behave when the server times out, API returns HTTP 4xx error, 5xx error).

The 'unittest 2.1' and 'responses 0.5.1' Python libraries are sufficient for properly testing, mocking and reporting city forecast comparisons. Refer to the Python test script (compare_forecasts_test.py) attached that automates the unit test cases and integration test.

Schedule

The work on the assignment needed to be done in less than a week, and the schedule for testing is estimated to be short with a single iteration. The test script reports properly, has about 10 tests that take about 5 minutes to perform in total. It should only take one test worker to run the unit tests script for testing the system and see test results matrix in the console.

Workflow diagram



Example test plan

Feature	Test cases	Expected result	Actual result
5 day city weather forecast	1st: null/ empty city name, Toronto Cleveland, Paris Cairo, Detroit Moscow Ottawa, unknowncityfoobar, 55$#	Validation exception, forecast for two cities, forecast for two cities, argument exception validation exception, validation exception	
Third party forecast service integration	1st: weather forecast HTTP request for 1 city, http request with bad APIKEY, http request with malformed city name, bad url request, mock request call when service is down	Get forecast response, Program validates http 4xx, Program validates 4xx error, Program validate 4xx error, Program validates http 5xx service unavailable	
Handling of city strings and forecast hours	1st: lower case upper case Toronto, forecast hour	Program validates strings and gets forecast comparison for the right cities at the right forecast time	

NOTE: Run the test cases in python compare_forecasts_test.py

Example Test scenario

Story <# and Name>, Iteration <#>, Tester <Who>

1) User requests 5 day weather forecast comparison for Toronto and Cleveland

Setup: Python installed and API service is available

Test scenario ID	Test scenario steps	Test scenario verification
1	• User U1 runs 'python compare_forecasts.py Toronto Cleveland' • User U1 chooses 9AM UTC forecast time from the command line • -OR- User U1 runs the first test from the test script compare_forecast_test.py	The program displays weather description and temperature (celsius) forecast comparisons between two cities for 5 days

Comments: ____________

Issues

The openweather 5 day forecast API sometimes has inaccurate functionality; for example passing a non-existent city name or a wrong city returns a weather response for a different city. Also, given the city and time of the request, the city weather forecast API response sometimes has weather information for only 4 days (not 5) at certain UTC forecast hours.