pylinkvalidator
Testing Plan : newAGEtech, Group H

Genevieve Okon (Okong), Abraham Omorogbe(Omorogoa),
Eric Le Fort(Leforte)

October 23, 2015

Contents

List of Figures

List of Tables

1 Revision History

Revision	Revision Date	Description of Change	Author
1	20-10-15	Initiate Test Plan Document	Genevieve Okon
2	20-10-15	Define Objectives,Scope, Constraints andTypes of tests	Genevieve Okon
3	20-10-15	Define Test Items	Abraham Omorogbe
4	20-10-15	Indicate project test factors	Abraham Omorogbe
5	20-10-15	Proofreading of test factors Creation test schedule and deliverables sections	Genevieve Okon
6	20-10-15	Plan for Automated Testing, Features to not be tested	Genevieve Okon
7	22-10-15	Proofreading and merging of overall content	Eric Le Fort
8	22-10-15	Correct table of contents mapping	Eric Le Fort
9	23-10-15	Modification to Factors to be tested	Abraham Omorogbe
10	23-10-15	Test factors rationale and testing methods	Abraham Omorogbe
11	22-10-15	Table of Contents	Eric Le Fort

Table 1: Revision History

2 Introduction

This is the test plan for the pylinkvalidator webcrawler application. It will address how the system will be tested; the approach, constraints, and schedule relating to the testing of the webcrawler. This document follows the Software Test Plan (STP) Template. This plan will address only those items and elements that are related to the process of producing an effective webcrawler which can traverses a web site, report errors encountered and also be able to download resources such as images, scripts and stylesheets. The details for levels of testing in this project will be addressed further in this test plan document, like in the testing approach section.

2.1 Objectives

This test plan for testing for the webcrawler program supports the following objectives:

1. To detail the activities required to prepare for and support the test.

2. To communicate to all responsible individuals the tasks which they are to perform and the schedule to be followed in performing the tasks.

3. To define the sources of information used to prepare the test plan.

4. To define what approach we expect to use for the tests and recommend and describe the testing strategies to be employed.

5. To define the test tools and environment needed to conduct the test.

6. To list the deliverable elements of the test activities.

7. To list the recommended test requirements.

8. Identify existing project information and the software components that should be tested.

2.2 Scope

This Test Plan applies to the unit, integration and system tests that will be conducted on the webcrawler. It presents a schema for testing the functionality of the webcrawler and the approach intended to be used to test this functionality. It has the objective to show that users of the webcrawler will be able to crawl websites in an easy and effective way, attempt to find missing attributes and failure conditions. This Revision 0 Test Plan serves to organize testing activities. This documentation at this level does not attempt to present much detailed testing information, as this information will be included in subsequent test reports as well as the final test plan.

2.3 Constraints

The principal constraint relating to the testing of this project is time. The project is to be delivered by December 2015, within this period we also need to focus on the development/implementation, documentation of the program as well as the testing. Human resources are also a constraint, as this project is limited to three individuals to perform all aspects of this project along with the consultancy of a supervisor.

2.4 Acronyms and Definitions

Table on next page

3 Tested Properties

3.1 Summary

These properties will be a succinct list of the aspects of the software we plan to test. The verification of the properties will be must useful in validating most non-functional requirements. A list of the properties, the explanation of each property and the different techniques we will implement to test the

Abbreviation	Meaning	Description
SDLC	Software Development Life Cycle	This describes a process for planning, creating, testing, maintaining and deploying a software project.
SRS	SRS Software Requirements Specification	Document reporting objectives, intended features, risks and constraints of the product. It is a comprehensive description of the intended purpose and environment for the software. The SRS fully describes what the software will do and how it will be expected to perform.
JUnit	Java unit test	Unit testing tool
PyUnit	Python unit test	Python language version of JUnit
PoC	Proof of Concept	Presentation with a purpose to demonstrate that the project is feasible
Broken link	Broken link	Is a link on a web page that doesn?t work.The destination website permanently moved or no longer exists
HTTP	Hypertext Transfer Protocol	HTTP is the protocol to exchange or transfer hypertext.
HTTPS	HTTP Secure	This is a protocol for secure communication over a computer network which is used on the Internet.
Anchor Tags	Anchor tags	If an anchor tag is within a webpage, links can be added to the body of the post which when clicked allow the user to jump to another location on the page.

Table 2: Acronyms and Definitions

properties are shown below:

The table 3 is a Test Properties List.

Test Components	Description	Method Used
Ease of Use	Generally difficulty and average time to input a query	Functional System Testing
Accuracy	Verify that specific output from the program is correct and accurate	Dynamic Testing
Speed	Verify the average speed of each query	Structural System Testing
Fault-Tolerance	Verify the program handle common users errors	Functional System testing
Adaptability	Verify program runs on multiple operating systems	Manual Testing
Legal Requirements	Verify web crawler cannot be used to crawler website, that discourage web crawls. i.e Kijiji and Footlocker	Functional System Testing

Table 3: Test Properties List

3.2 Properties To Be Tested on Pyweblinkvalidator

Different properties of the program that must be considered and verified through testing.

3.2.1 Ease of Use

The main objective of ease of use testing is ensuring the program is easy to set up and can easily be used by the stakeholders mentioned in the SRS document.

Rationale

This program setup time and ease of use must be verify. As developers we need to ensure the program is easy enough for any of the stakeholders to use and setup in minutes.

Testing Approach

We are going to use one of the Functional System Tests, Usability Testing. To prove our program is easy to use and step up. We will analysis our code and make sure everything is well documented, users are prompt corrected and the program is fairly easy to debug. After we believe the code is sound and easy to use, we will ask 5 non-engineers to use the program to crawl their favourite website and they will rate the ease of use. We aim to have an average score of 8/10

3.2.2 Accuracy

The main objective for the accuracy testing is to verify the results and status codes from the web-crawler are accurate.

Rationale

This program must have accurate results, so customers and stakeholders trust to products and will continue to use it to increase efficiency.

Testing Approach

We are going to use one of the Dynamic Testing. To prove our program is produces the correct results we will have a set of test HTML files that the program shall crawl. The programs output must correspond to what occurs when a user manual clicks all the links on the test HTML and the status codes on each HTML page.

3.2.3 Speed

The main objective of speed testing is to make sure the program, returns results in a timely manner.

Rationale

This program speed must be verified, we must assure stakeholders that the program will work quickly and increase efficiency, like we mentioned in our SRS.

Testing Approach

We are going to use one of the Structural System Tests, Speed Testing. To prove the program will run in a reasonable timeframe, In order to do this we are going to setup a stopwatch function at the begin of each query and it will print the time it took for each query. The time to crawl a page for any content should be between 0-2 seconds.

3.2.4 Fault-Tolerance

The main objective of Fault-Tolerance is to make sure this program can handle any kind of user input

Rationale

This program should able to take any input and if the input is the wrong format, the program should inform the user or fix the issue. This feedback with be essential in helping user use the software correctly.

Testing Approach

We are going to use one of the Functional System Tests, Error Handling. To prove the program can handle invalid entries and it can correct inputs missing HTTP:// or www before the domain name,

we with input queries that should break the program. Queries include but not limited to 23wew, example.com, www.hi.me,1/2/2 and much more. These queries should be handled with such response as "invalid entry" or a list crawled website (normal output) if the domain was missing HTTP:// or www.

3.2.5 Adaptability

The main objective of Adaptability is to make sure this program will work as expect on any operating system.

Rationale

This program should be platform independent, our software should be able to run on any system that the use decides to use.

Testing Approach

We are going to use Manual Tests to verify this works. The team with test is the program works on a Linus system, Mac system and Windows system. We will load up the program on each of the mentioned Operating Systems and verify the program acts identical across each platform.

3.2.6 Legal Requirements

The main objective of Legal Requirements, verify our program follows the law and a website's Terms of Use.

Rationale

This program should block sites that has strict rules against web crawl, such as Kijiji, Foot Looker and eBay

Testing Approach

We are going to use Dynamic Testing. We are going to query website on our list of forbidden website and make sure the program respond with a message along the lines of "Web Crawling is not permitted by this website". We will query sites on our forbidden list such as Kijiji and eBay and verify the program outputs the message above.

4 Tested Components

4.1 Summary

These components are different parts of our code and design we need to test and review in order verify all functional requirements have been met and the program is functioning correctly. The following section

with be a list of all the components we plan on testing and an overview of how we plan to test these components.

1. Software Requirements Specifications
2. Code
3. Features
4. Error Handling

4.2 Software Requirements Testing

As we continue working on this project and developing more features, we most be able to run some tests that can verify all the requirements in the SRS have been met. We will implement black box tests to achieve this.

4.3 Code Testing

As we continue to build we must vet every team members commits to the repository. After every submission any team member other than the original submitter should analyze the submitted code, and make sure all the is sound logically and functionally. We would be using a form white box testing.

4.4 Features Testing

We will be observing the code as we add more features. We must verify every feature does exactly what it is supposed to do and each feature can be traced back to a requirement. More details on individual test cases with be provides sections 5 and 6.

4.5 Error Handling Testing

We are building a program that should be able to handle any query from the user and different error that may occur during run-time. We will implement tests that will reproduce these errors and ensure there are handled corrected. More details mentioned in sections 3,5 and 6.

5 Features Tested in POC

5.1 Webpage HTML File Download

This application needs to be able to download the HTML file associated with a webpage. It should download the whole file without any missing or incomplete data.

Approach

This feature will be tested by setting up sample webpages or using existing ones on the web and verifying the data received by this application matches that stored in the file. This test will be considered

to be passed if all expected results occur.

Test Cases (Blackbox, Using PyUnit)
The process will involve reading in the files listed below and then comparing the results to the expected results.
The output will be a String containing the data read in.

Files Used:

- a.html
- alone.html
- c.html
- d.html
- f.html
- index.html

5.2 Find Anchor Tags in HTML file

This application needs to parse through an HTML document in order to locate all of the links to other pages. This is what the application will need to do in order to continue crawling deeper into a website.

Approach
This feature will be tested by setting up sample webpages or using existing ones on the web and verifying that all link tags are extracted correctly. This test will be considered to be passed if all expected results occur.

Test Cases (Blackbox, Using PyUnit)
The process will involve parsing a file, extracting the links, comparing them to expected links.
The output will be an array of Strings containing the extracted links.

Files Used:

- a.html
- alone.html
- c.html
- d.html
- f.html

6 Features Tested in Final

6.1 Depth First Crawl

This type of crawl involves going down into every link that is found recursively down as deep as possible first. This is optimal for smaller websites as you can analyze the whole website very efficiently.

Approach

This feature will be tested by setting up a sample website and ensuring all of its pages are traversed in the correct order as well as that pages that shouldn?t be accessible are not. This test will be considered to be passed if all expected results occur.

Test Cases (White Box, Using PyUnit)

root.html, (perform algorithm, record links visited and any notifications), (String array containing the list of links and notifications in the order they were visited) evenTree.html

6.2 Breadth First Crawl

This type of crawl involves finding every link found on a page and then following each one. From there, this step occurs iteratively while there are remaining links that don?t lead to dead-ends. This is optimal for larger websites as you can analyze the website layer by layer in a short time instead of taking a large amount of time delving all the way down first.

Approach

This feature will be tested by setting up a sample website and ensuring all of its pages are traversed in the correct order as well as that pages that shouldn?t be accessible are not. This test will be considered to be passed if all expected results occur.

Test Cases (White Box, Using PyUnit)

root.html, (Perform algorithm, record links visited and any notifications), (String array containing the list of links and notifications in the order they were visited) evenTree.html

6.3 Download Media Resources

This application should be able to download various resources that are made readily available on the host server. Notifications will also be provided for any resources that are not visible or that are not available for download.

Approach

This feature will be tested by adding various resources to our website (including broken links) and attempting to download them. We will then test to ensure that the downloaded resource matches the original version or that broken links return the correct notification. This test will be considered to be passed if all expected results occur.

Test Cases (White Box, Using PyUnit)

media.html, (Performs a crawl, downloads all the images on each page visited), (Nested folders containing all the downloaded images for each page (i.e. img1, img2, etc)) evenTree.html

6.4 Error Verification and Notification

This application must inform the user of various issued raised while performing its checks. These errors include various HTML error codes such as server faults, permission conflicts and invalid requests.

Approach

This feature will be tested by forcing the occurrence of errors and ensuring that the notifications that are provided match the ones expected to. This test will be considered to be passed if all expected results occur.

Test Cases (White Box, Using PyUnit)

errorLinks.html, (Parse file, handle all broken links correctly, receive error notifications), (String array containing the notifications in the order that they occurred)

6.5 Attempting to Correct Invalid Links

In HTML code, links are frequently written without the ?http://? or ?www? part of their prefix which will bring rise to errors when attempting to connect to the webpage. This application will attempt to correct this issue by adding the to any broken links before creating any error notifications.

Approach
This feature will be tested by providing broken links, functioning links and links that are just missing the prefix. The application should do nothing to the functioning links and try to append the prefix to the links that did not work. The broken links should still not work, the functioning links should still be functioning and those links missing their prefix should now be functioning. This test will be considered to be passed if all these expected results occur.

Test Cases (Black Box, Using PyUnit)

www.google.ca, (check for http://, add http://), (http://www.google.ca) htttp://www.google.ca, (check for http://, try adding http:// and http://www.), (Error in link notification) google.ca, (check for http:/ try adding http:// and http://www.), (http://www.google.ca) http://google.ca, (check for http://, add www. after), (http://www.google.ca)

6.6 Recognizing Cycles within the Links

When traversing a website, it is possible to have links which end up creating a cycle. A computer would not recognize this and would end up stuck traversing this loop infinitely. This application should be able to keep track of previously visited links and compare them to new ones in order to prevent this situation.

Approach

This feature will be tested by setting up sample webpages that contain cycles within the structure of their links. The program should be able to avoid getting caught in these cycles. This test will be considered to be passed if all tests perform the expected number of steps to completely traverse the websites.

Test Cases (Using PyUnit)

cycle0.html, (Enters the file, records address name, continues visiting addresses, doesn?t enter a previously visited address), (The list of three files)

6.7 Usability Testing

This application must be usable from anyone with basic terminal usage experience with how to run a python application. There should not be much of a learning curve, at least for the more simple functionality of the program.

Approach

The usability metric will be determined by getting a sample of ten individuals who are capable of running an application from the Terminal or Command Line. These users will perform a series of predetermined tasks using the program and then fill out a brief survey regarding their experience. This test will be considered to be passed if the average scoring of the survey is 90% or higher.

Tested by way of a survey.

6.8 Reliability Testing

This application must reliably produce the same (correct) results for the same inputs. For example, if there is no change in the state of any pages on the website, performing a traversal should output the same links and downloading resources should return the same resources.

Approach

The aforementioned individuals will be required to perform certain tests multiple times in order to confirm that the results are consistent. This test will be considered to be passed if the average score of these tests are 90% or higher (100% would be optimal but it is possible the error is on the tester?s end).

Tested by way of a survey.

7 Features Not Tested

No features will go untested.

8 Automated Testing Plans

8.1 PyUnit

Unit and integration tests will be used to test this webcrawler. PyUnit is Python's unit testing framework. Unittest supports test automation, it also supports some important concepts like test fixture, test suite, test runner and test case. A test case is the smallest unit of testing. It checks for a specific response to a particular set of inputs. These tests can be used to check if the results of a webcrawler match an expected result. e.g. The error code for a particular website.

9 Testing Schedule

Testing will be performed at several points of the SDLC. The test report revision 0 will be completed by November 23 2014 and the final test report will be completed by December 8 2015. The group members and the supervisor will handle the organization of task division and timelines for the project completion. The python unit tests on the webcrawler will be performed every week, during these tests any errors found in the webcrawler implementation and states will be corrected. The test cases will be modified every month since the development of the webcrawler will be progressed because of new features implemented. Also before any code for the program is committed it is the individual?s responsibility to test it.

10 Appendix

10.1 Test Deliverables

10.2 Files to be Used During Testing

- evenTree.html: This html page has 2 links. Each of these links point to 4 links to other pages.
- brokenLinks.html: This html page has a few broken links and not much else.
- cycle0.html: This html page links to cycle1.html, cycle1.html links to cycle2.html and cycle2.html links back to cycle0.html to complete a cycle.

Test Document	Objective	Delivery date
Test Plan - Revision 0	Set some deliverables for future test cases. Set initial guidelines and objectives for testing	October 19, 2015
Test Report -Revision 0	Report tracking progress, history and results of tests performed	November 23, 2015
Test Plan -Final	Indicate test cases and considerations for testing prior to the final completion of the webcrawler program	December 8, 2015
Test Report - Final	Report tracking progress, results of test and whether they were expected or not and history of tests performed	December 8, 2015

Table 4: Test Deliverables