SE 3XA3: Test Report
Google Images Downloader

Team 201, CAS Dream Team
Sam Crawford, crawfs1, 400129435
Joshua Guinness, guinnesj, 400134735
Nicholas Mari, marin, 400132494

April 6, 2020

Contents

List of Tables

List of Figures

Table 1: **Revision History**

Date	Version	Notes
04/05/2020	1.0	First Draft of Test Plan
04/06/2020	1.1	Nonfunctional Test Summary Added
04/06/2020	1.2	Rest of sections complete

This document is intended to summarize the results of all tests on the system, while indicating the changes made to the system due to issues discovered during said testing.

1 Functional Requirements Evaluation

Test Name: FR-SQ1
Results: The tester was able to confirm that ten images relating to McMaster existed at google-images-downloader/src/Images/McMaster within 15 seconds of running the command.

Test Name: FR-SQ2
Results: The automated test passed meaning that a folder existed at google-images-downloader/src/Images/default that contained three images with the file extension .gif

Test Name: FR-SQ3
Results: The automated test passed meaning that a folder existed at google-images-downloader/src/Images/default that contained three images with the file extension .jpg

Test Name: FR-SQ4
Results: The automated test passed meaning that a folder existed at google-images-downloader/src/Images/default that contained three images with the file extension .png

Test Name: FR-SQ5
Results: The automated test passed meaning that within the list of URLs to

download, “mcmaster.ca” was not present.

Test Name: FR-SQ6
Results: The automated test passed meaning that within the list of URLs to download, “mcmaster.ca” was present in every URL.

Test Name: FR-SQ8
Results: The automated test passed meaning that a folder existed at google-images-downloader/src/Images/homestuck that contained five images with a size of >2MP.

Test Name: FR-SQ9
Results: The automated test passed meaning that the list of URLs generated from the json output, and the list generated from the files are equal to each other.

Test Name: FR-SQ10
Results: The tester was able to confirm that a folder existed at google-images-downloader/src/Images/fruit that contained three images which are red in color.

Test Name: FR-SQ11
Results: The tester was able to confirm that a folder existed at google-images-downloader/src/Images/waterfall that contained three panoramic images.

Test Name: FR-DL1
Results: The automated test passed meaning that a folder existed at google-images-downloader/src/Images/software that contained forty images.

Test Name: FR-DL2
Results: The automated test passed meaning that a folder existed at google-images-downloader/src/newDirec/software that contained two images.

Test Name: FR-DL3
Results: The tester confirmed that a folder existed at moore.mcmaster.ca:

/u50/guinnesj/McMaster that contained three images within 15 seconds.

2 Nonfunctional Requirements Evaluation

2.1 Look and Feel

Test Name: NFR-AR1
Results: Throughout all of the testing, the output messages displayed to the console were always visible. Running on Windows, specifically with the MinGW environment, caused all of the messages to be printed all at once when the program has concluded its execution, but this issue is tied to how Python buffers its output, and not to the program itself. This anomaly likely exists on some other platforms and configurations as well.

2.2 Usability

Test Name: NFR-EUR1
Throughout the testing process, all intended correct executions of the program caused the desired images to be download. In addition, running the program with incorrect or with no arguments causes the proper usage to be displayed to the user, informing them of the correct way to run the program.

Test Name: NFR-ELR2
Results: Running the commands specified in the test plan clones the git repository and successfully downloads all required libraries to the user's system.

2.3 Performance

Test Name: NFR-RR1
Results: Running the program with incorrect or with no arguments causes the proper usage to be displayed to the user, informing them of the correct way to run the program.

2.4 Operational and Environmental

Test Name: NFR-IAR1
Results: Running the program on the most recent version of software and

with the most recent version of the Google Images software results in the expected output and the correct downloaded images.

2.5 Maintainability and Support

Test Name: NFR-ADR1
Results: The system works as expected on Linux, Windows, and Mac systems. The program detects the platform of the system it's run on and chooses the correct Chrome driver, located in the downloaded repository itself, as well as the location of the Chrome binary file on the user's system.

2.6 Legal

Test Name: NFR-SR1
Results: Running the flake8 linter, with specific warnings suppressed when they conflict with the coding style chosen by the team (as specified in the project's Development Plan, returns no issues or suggestions to improve the style of the source code.

3 Comparison to Existing Implementation

The implementation of the project as defined by this test report differs in many regards to the original it was based on. The most obvious difference is that since this project began, Google had changed their method of storing image URLs in the web page for search results. Instead of storing the URLs directly in the HTML, the URLs were resolved dynamically in the JavaScript, only being present in the HTML when the image was clicked on. Because of this, the original implementation does not work. This problem was circumvented in this implementation by using selenium to navigate the page, "clicking" on each image, so that its URL could be scraped.

Another key difference to the original is that the new implementation, while still written in Python, is split into modules, while the original implementation was monolithic. The division of the program into modules supports the proper software engineering principles of "Separation of Concerns" by ensuring that each module contains one secret, so that the internal implementation is separate from the external interface. This means that no

matter if or how the underlying code changes, the program as a whole will still run properly, as long as the external interface is respected and upheld.

The original implementation did not have the functionality of the new one to move the downloaded files to a server. This feature is applicable to a user who wishes to download images and then upload them to a server, perhaps to use for machine learning.

Another feature added to the redesign is the ability to read the input parameters from a file, using the `fromfile` parameter. The original allowed for the keywords to be read from a file, but by allowing the entirety of the input parameters to be stored in a file leads to a larger potential for automation of image downloading.

Due to the time constraints of the project, there are some features that the original implementation, when it worked, had that the re-implementation does not. The original implementation had a "silent mode" that would run the program without printing anything to the console. There were also some flags that the redesign is missing. Some examples are `related_images`, which also retrieves related images, `extract_metadata`, which allows the metadata of the downloaded images to be stored as a .json file, and `no_download`, which only retrieves the image URLs without downloading them.

The original implementation included the parameters `prefix_keywords` and `suffix_keywords`, which allowed the user to input a set of prefixes and a set of suffixes which would be built with the keyword. For example, if the prefix list was {"software", "materials"} and the suffix list was {"graphs", "labs"}, running the program with the keyword "engineering" would search for "software engineering graphs", "software engineering labs", "materials engineering graphs", and "materials engineering labs". However, this was implemented with a triple `for` loop in the original, which is not optimal at all. The design decision was made to exclude this functionality altogether from the re-implementation, since the ability to read the parameters from a text file leads to easy automation.

4 Unit Testing

All of the modules except for Main.py were unit tested. Main.py was not unit tested as it solely facilitates the flow of data through the system and its functionality is tested through integration and system tests.

Within each module, almost all functions are directly tested, and those

that are not are indirectly tested through testing other functions. The types of unit testing are grey box which are more focused on program logic and the flow of data through the function. All the unit tests were implemented with pytest and successfully passed when they are run.

All testing files can be found under the testing folder located at: google-images-downloader/src/Test/. The ones named <ModuleName>Test.py contain the unit tests. The name of each unit test gives a description of what is being tested.

5 Changes Due to Testing

There were many instances of changes made to the system as a result of the testing described in this report.

The first major change was made to the Input module with the function used to get the components from a specified configuration file. Through testing, it was discovered that there was an error that arose when trying to parse the configuration file into a dictionary. This error arose when a configuration file was missing certain parameters and then when parsed into a dictionary would cause issues when the system would try and access the missing parameter. This resulted in an overhaul of the error handling of this function allowing the system to still run if a parameter, excluding keyword, was missing.

The assumption was made that image URLs that don't contain the extension (such as those hosted on the S3 storage service) couldn't be downloaded. However, during the testing of the Output module, it was discovered that the file type data, while not present in the URL itself, is still stored within the data of the image that would be read and stored to be processed. This modification allowed for these types of images to be downloaded instead of throwing an error, which increased the effectiveness of the program.

While verifying that the proper arguments are displayed upon an improper execution of the program, a tester noticed that countries with accents (Côte d'Ivoire and São Tomé and Príncipe) were improperly rendered as distorted Unicode boxes in some terminals. To reduce ambiguity and increase usability, the decision was made to remove the accent from the textual representation of these countries ("Cote d'Ivoire and Sao Tome and Principe"). Although this change might upset some users of the program, or lead to confusion when entering the accented version of the country name doesn't work,

the team agreed that the benefits outweighed the pitfalls.

6 Automated Testing

Wherever possible, testing was automated to allow for the creation of a test suite that could be run whenever a change was made to ensure that the functionality of the the program was not compromised. Most of this automated testing involves checking that images have actually been downloaded, and then checking various properties of the files. Some tests check that the file type is correct, while others check the image size via the image's dimensions. Some flags, like `colour`, had to be verified manually, by checking that all downloaded images were green, for example. Others, like `safesearch` or `region`, have no artifacts present in the downloaded images, because the metadata isn't stored with the image, and the actual filtering of images to make sure that the images match the given criteria is implemented at Google's end.

7 Trace to Requirements

Test Id	**Requirement**						
	AR1	EUR1	LR1	RR1	IAR1	ADR1	SR1
NFR-AR1	✓						
NFR-EUR1		✓					
NFR-LR1			✓				
NFR-RR1				✓			
NFR-IAR1					✓		
NFR-ADR1						✓	
NFR-SR1							✓

Table 2: **Nonfunctional Requirements to Tests Traceabillity Matrix**

Test Id	Requirement														
	FR1	FR2	FR3	FR4	FR5	FR6	FR7	FR8	FR9	FR10	FR11	FR12	FR13	FR14	FR15
FR-SQ1	✓														
FR-SQ2				✓											
FR-SQ3				✓											
FR-SQ4				✓											
FR-SQ5					✓										
FR-SQ6						✓									
FR-SQ8								✓							
FR-SQ9									✓						
FR-SQ10							✓								
FR-SQ11											✓				
FR-DL1		✓													
FR-DL2			✓												
FR-DL3										✓					

Table 3: **Functional Requirements to Tests Traceabillity Matrix**

8 Trace to Modules

Test Id	**Requirement**					
	Main	Input	SearchQuery	NavigatePage	Output	HardwareHiding
FR-SQ1	✓	✓	✓	✓	✓	✓
FR-SQ2	✓	✓	✓	✓	✓	✓
FR-SQ3	✓	✓	✓	✓	✓	✓
FR-SQ4	✓	✓	✓	✓	✓	✓
FR-SQ5	✓	✓	✓	✓		
FR-SQ6	✓	✓	✓	✓		
FR-SQ8	✓	✓	✓	✓	✓	✓
FR-SQ9	✓	✓	✓	✓		✓
FR-SQ10	✓	✓	✓	✓	✓	✓
FR-SQ11	✓	✓	✓	✓	✓	✓
FR-DL1	✓	✓	✓	✓	✓	✓
FR-DL2	✓	✓	✓	✓	✓	✓
FR-DL3	✓	✓	✓	✓	✓	✓

Table 4: **Modules to Tests Traceabillity Matrix**

9 Code Coverage Metrics

The implemented test cases account for 92% branch coverage and 93% statement coverage. The test plan calls for 80% coverage in both areas, so the test suite for the program exceeds its expectations. These numbers were obtained using the pytest-cov tool. Test cases were designed to account for normal usage, as well as exception and boundary cases.

Coverage report: 92%

Module ↓	statements	missing	excluded	branches	partial	coverage
Input.py	53	1	0	16	1	97%
Main.py	13	7	0	4	1	41%
NavigatePage.py	54	6	0	16	2	86%
Output.py	102	18	0	20	3	83%
SearchQuery.py	18	0	0	6	0	100%
Test/InputTest.py	37	0	0	0	0	100%
Test/IntegrationTest.py	32	0	0	2	0	100%
Test/NavigatePageTest.py	30	0	0	4	0	100%
Test/OutputTest.py	109	0	0	0	0	100%
Test/SearchQueryTest.py	26	0	0	0	0	100%
Test/systemTest.py	91	5	0	12	0	93%
Total	**565**	**37**	**0**	**80**	**7**	**92%**

coverage.py v5.0.4, created at 2020-04-05 21:44

Figure 1: Branch Coverage Results

Coverage report: 93%

Module ↓	statements	missing	excluded	coverage
Input.py	53	1	0	98%
Main.py	13	7	0	46%
NavigatePage.py	54	6	0	89%
Output.py	102	18	0	82%
SearchQuery.py	18	0	0	100%
Test/InputTest.py	37	0	0	100%
Test/IntegrationTest.py	32	0	0	100%
Test/NavigatePageTest.py	30	0	0	100%
Test/OutputTest.py	109	0	0	100%
Test/SearchQueryTest.py	26	0	0	100%
Test/systemTest.py	91	5	0	95%
Total	**565**	**37**	**0**	**93%**

coverage.py v5.0.4, created at 2020-04-05 21:57

Figure 2: Statement Coverage Results