SE 3XA3: Test Plan
Google Images Downloader

Team #201, CAS Dream Team
Nicholas Mari, marin
Samuel Crawford, crawfs1
Joshua Guinness, guinnesj

April 5, 2020

Contents

List of Tables

Date	Name	Version	Notes
2/26/2020	Nick	1.0	Created Document
2/27/2020	Sam	1.1	Filled in Tests for Nonfunctional Requirements
2/27/2020	Josh	1.2	Filled in Test Plan and started Unit Test Plan section
2/27/2020	Nick	1.2.1	Started Tests for Functional Requirements
2/27/2020	Sam	1.3	Filled in Purpose and Overview
2/28/2020	Sam	1.4	Filled in Scope
2/28/2020	Josh	1.5	Completed Tests for Proof of Concept and Unit Testing Plan sections
2/28/2020	Nick	1.6	Completed Tests for Functional Requirements and Traceability Matrices
2/28/2020	Sam	1.6.1	Minor consistency and formatting improvements
4/05/2020	Joshua	1.7	Rev1 Changes:

Table 1: **Revision History**

1 General Information

1.1 Purpose

The purpose of this document is to provide an overview of the testing, validation, and verification of google-images-downloader. These test cases were designed to be used as a reference during the implementation of both the program's source code and its actual testing.

1.2 Scope

The scope of the testing of google-images-downloader will ensure that all testable requirements, both functional and nonfunctional, are met and that the problems found in the proof of concept are addressed. It will also confirm that each distinct part works in isolation and with its collaborators, and that the system as a whole performs its duties as expected.

1.3 Acronyms and Definitions

Abbreviation	**Definition**
UTF-8	Unicode Transformation Format - 8-bit

Table 2: **Table of Acronyms**

Term	**Definition**
byte encoding	The process of encoding data into a series of bytes.
Gantt chart	A bar chart that illustrates a project schedule.

Table 3: **Table of Definitions**

1.4 Overview of Document

This document outlines the testing plan for google-images-downloader. This includes a description of the software, the team of testers, the approach to automated testing, and the tools and schedule used to optimize the testing

efficiency and accuracy. This document also includes a description of the testing required for the system, and outlines the various tests for both the functional and nonfunctional requirements. This will also cover the traceability between the test cases and the requirements. Other tests outlined in this document are those that arose from the proof of concept, and those for testing each individual unit of the system.

2 Plan

This section will give an overview of the testing plan, including but not limited to tools used, team, and approach.

2.1 Software Description

The software system is a google images downloader command line tool that will allow end users to download a certain number of googles images given specific keywords. The aim of this tool is to provide assistance to those involved in machine learning, and secondarily those involved in art.

The system will consist of two main subsystems. The first subsystem will be responsible for taking in user input, and constructing the google images query. The second system will be responsible for downloading the images from google and storing them.

2.2 Test Team

The test team will consist of the three developers working on the project, Joshua Guinness, Sam Crawford, and Nicholas Mari.

2.3 Automated Testing Approach

Besides simple exploratory testing done during development for quick confirmation, the bulk of testing effort will be concentrated on developing three types of automated tests: unit, integration, and system. Each of these can be ~~through~~ thought of as having a different level of abstraction with unit testing at the lowest level and system testing at the highest level. All of these will be written in pytest, with a separate testing file for each type to

provide greater clarity. The three main aspects of the system being tested are simulated user input, program flow, and program logic.

Unit tests ensure functions behave according to their design. Unit testing will also focus on boundary cases. These will be written after each module is implemented to ensure correctness.

Integration tests provide assurance that modules work together correctly and they interface properly. These will be done after a group of similar modules is complete, and after a sub-system is complete. They will also be done between sub-systems after the implementation is almost done.

Finally, system testing is concerned with ensuring that the final program adheres to the functional and non-functional requirements. This will be done after almost all of the system is complete.

Since these tests will be written in pytest, they will be run as a regression suite before a developer commits their code to GitLab. Pytest is a command line tool that will be run locally by each developer.

2.4 Testing Tools

One of the tools to be used is pytest, a unit testing framework for python. Pytest is a standalone library that will be used implementing and running the unit tests.

The other tool is pytest-cov a plugin that produces coverage metrics using pytest.

The linter flake8 will be used as static code quality analysis tool during development.

2.5 Testing Schedule

See Gantt Chart here for the testing schedule.

Unit testing will occur parallel to development. See Section 5 for further details on that. Integration testing and system testing will be done after development. All three levels of testing are visualized on the Gantt chart.

3 System Test Description

3.1 Tests for Functional Requirements

3.1.1 Search Queries

FR-SQ1: **Type:** Functional, Dynamic, Manual

Initial State: An empty download Repository

Input: A search request with keyword "McMaster"

Output: Images downloaded all related to McMaster

How test will be performed: The tester will run the program to download the images and manually check if the images downloaded match the keyword given.

FR-SQ2: **Type:** Functional, Dynamic, Automatic

Initial State: An empty download repository

Input: A search request with only a specific file type requested

Output: A download repository with only files of the specified type downloaded

How test will be performed: An automated test case will send the request to the system and check that the file extensions on each of the files downloaded are the specified type.

FR-SQ3: **Type:** Functional, Dynamic, Manual

Initial State: An empty download repository

Input: A search query with a request to not download from a specified website

Output: A download repository full with images, with no images from the blacklisted website

How test will be performed: The tester will manually start the search query with all the specified inputs and check each downloaded image URL to ensure that it does not come from the blacklisted website.

FR-SQ4: **Type:** Functional, Dynamic, Manual

Initial State: An empty download repository

Input: A search query with a request to only download from a specified website

Output: A download repository full with images, with no images from the blacklisted website

How test will be performed: The tester will manually start the search query with all the specified inputs and check each downloaded image URL to ensure that it comes from the white-listed website.

FR-SQ5: **Type:** Functional, Dynamic, Manual

Initial State: An empty download repository

Input: A search request with prefixes and suffixes given

Output: A download repository filled with images that match the search query and prefix or suffix

How test will be performed: The tester will manually ensure that each image matches both the search query and the prefixes and suffixes given.

FR-SQ6: **Type:** Functional, Dynamic, Automatic

Initial State: An empty download repository

Input: A search request with the size of desired images specified

Output: A download repository filled with images of the specified size

How test will be performed: An automated test case will check the size of each image to ensure it matches the size specified in the search request.

FR-SQ7: **Type:** Functional, Dynamic, Manual

Initial State: An empty download repository

Input: A search request with all constraints and keywords in a configuration file provided to the system when the user makes the request

Output: Download repository filled with images

How test will be performed: The tester will manually ensure that all the images downloaded match the keyword and search constraints given in the configuration file.

3.1.2 Download Images

FR-DL1: **Type:** Functional, Dynamic, Automatic

Initial State: An empty download repository

Input: A search request with a limit of 50 images specified

Output: A download repository contains 50 images in it

How test will be performed: An automatic test case will run the program with the specified input and count the number of new files in the download repository to ensure it is correct.

FR-DL2: **Type:** Functional, Dynamic, Automatic

Initial State: An empty download repository

Input: A download request with a specified download folder

Output: Images downloaded to the specified folder

How test will be performed: An automated test case will request a search with the specified parameters and automatically check to ensure that the images were downloaded to the folder the test case specified in the search.

3.2 Tests for Nonfunctional Requirements

3.2.1 Look and Feel Requirements

Appearance Requirements

NFR-AR1: **Type:** Functional, Dynamic, Manual, Acceptance

Initial State: A clean version of the program

Input/Condition: A typical usage of the system

Output/Result: Output messages displayed on the console

How test will be performed: The tester will use a survey to verify that at least 80% of users find that the output messages fit the console nicely.

3.2.2 Usability and Humanity Requirements

Ease of Use Requirements

NFR-EUR1: **Type:** Functional, Dynamic, Manual, Acceptance

Initial State: A clean version of the program

Input/Condition: A typical usage of the system

Output/Result: A folder of the desired downloaded images

How test will be performed: The tester will use a survey to verify that at least 90% of users are able to successfully download their images after reading the sample inputs and explanations for the first time.

Learning Requirements

NFR-LR2: **Type:** Structural, Static, Manual, Acceptance

Initial State: A computer without the system installed

Input/Condition: A typical installation of the system from the command line

Output/Result: A folder of the desired downloaded images

How test will be performed: The tester will use a survey to verify that at least 90% of users that are comfortable using the command line are able to download the program within five minutes.

3.2.3 Performance Requirements

Robustness Requirements

NFR-RR1: **Type:** Functional, Dynamic, Manual, Acceptance

Initial State: A clean version of the program

Input/Condition: An invalid input; an input that would result in an error

Output/Result: A descriptive error message

How test will be performed: The tester will use a survey to verify that at least 75% of users can identify their mistake from the error message.

3.2.4 Operational and Environmental Requirements

Requirements for Interacting with Adjacent Systems

NFR-IAR1: **Type:** Functional, Dynamic, Manual

Initial State: A clean version of the program installed on a computer with the latest version of the Google Images HTML return code.

Input/Condition: A typical usage of the program

Output/Result: A folder of the desired downloaded images

How test will be performed: The tester will manually verify that the program can download all the images using the latest format.

3.2.5 Maintainability and Support Requirements

Adaptability Requirements

NFR-ADR1: **Type:** Functional, Dynamic, Manual

Initial State: A clean version of the program on a Linux, Windows, and Mac platform.

Input/Condition: A typical usage of the program

Output/Result: A folder of the desired downloaded images

How test will be performed: The tester will manually verify that at all test cases pass on all platforms.

3.2.6 Legal Requirements

Standards Requirements

NFR-SR1: **Type:** Structural, Static, Manual

Initial State: A clean version of the program

Input/Condition: An execution of the linter

Output/Result: A successful linting with no errors to report

How test will be performed: The tester will perform a code review to manually verify that the code matches Google's coding style and that the linter returns no errors.

3.3 Traceability Between Test Cases and Requirements

Test Id	**Requirement**								
	FR1	FR2	FR3	FR4	FR5	FR6	FR7	FR8	FR9
FR-SQ1	✓								
FR-SQ2				✓					
FR-SQ3					✓				
FR-SQ4						✓			
FR-SQ5							✓		
FR-SQ6								✓	
FR-SQ7									✓
FR-DL1		✓							
FR-DL2			✓						

Table 4: **Functional Requirements Traceabillity Matrix**

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 5: **Nonfunctional Requirements Traceabillity Matrix**

The following Nonfunctional Requirements are not included in the Traceabillity Matrix as they do not have a test case associated with them:

- ~~LR2~~
- PDR1: Simply a true or false question of whether the git repository is public.

- MSR1: Concerns a future development strategy therefore cannot be tested.
- MSR2: Concerns a future development strategy therefore cannot be tested.
- ACR1: Simply a true or false question of whether the git repository is public.
- CR1: Impossible to confirm with certainty but best efforts were made by developers.
- CPR1: Simply a true or false question of whether the README.md contains the statement.

4 Tests for Proof of Concept

There were two main issues that ~~we ran into~~ occurred while implementing the proof of concept that required significant time to solve.

The first is that the Google Images HTML code structure changed from when the original project was first confirmed to run properly, and the start of implementing the proof of concept. This meant that the section of the original project regarding how the HTML is scrapped and the images downloaded was almost useless. To solve the issue, ~~we first figured out what~~ the new HTML structure was first figured out by using "Inspect Element" on ~~images.~~ the page of google image results. This also took some time as there are multiple links within each image subsection. Once ~~we~~ it was figured out where the image was stored in the HTML, ~~we could then scrape~~ it could then be scraped and parsed ~~it~~ to download it.

The second issue occurred after downloading the raw HTML code and trying to parse it for the image links. It would not parse properly, even after it was confirmed that the HTML downloaded contained the links or phrases that were being searched for. ~~We solved the issue~~ The issue was solved by going through each step of the code slowly and printing things out to the console. ~~We~~ It was discovered that the raw HTML downloaded was actually byte encoded, as evidence by the "b" in front of the string. Adding a statement to decode it to UTF-8 solved the issue.

5 Unit Testing Plan

This section will go over the unit testing plan for this project including the plan for the testing of internal functions, and output files. The tool pytest will be used for all unit testing.

5.1 Unit testing of internal functions

The purpose of unit testing is to ensure that a specific unit of software performs as intended. On a software function level, this is done by sending an input to a function, and asserting that the output is the expected output of the function.

For this project, after every module is complete, it will be unit tested before the developer starts work on the next module. This means that unit testing will be done during the development process, not all at the end. The developers can then work from a place of confidence and assurance that previous code written functions as intended.

The types of unit tests implemented by the developer will be grey box tests. Grey box tests are a combination of white box and black box where the goal is to exploit knowledge of the system to test interfaces and internal functionality

The coverage metrics we will be using are 85% statement coverage and 80% branch coverage. This data will be found using the pytest-cov tool. There are no stubs or drivers needed for testing.

5.2 Unit testing of output files

The only output files produced are the downloaded images themselves. The contents of the file will not be unit tested at all; unit testing will revolve around the properties (eg. ~~name, aspect ratio, bit-depth~~ file type, size) of the images and the number of images downloaded.

6 Appendix

6.1 Symbolic Parameters

The test cases listed here do not call for the use of any symbolic constants.

6.2 Usability Survey Questions

In addition to relevant questions for testing the appropriate nonfunctional requirements (see Sections 3.2.1, 3.2.2, and 3.2.3), users will also be asked to confirm that no part of the product (eg. naming conventions) are offensive to them.