

# SE 3XA3: Test Plan Google Images Downloader

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

April 6, 2020

# Contents

<b>1</b>	<b>General Information</b>	<b>1</b>
1.1	Purpose . . . . .	1
1.2	Scope . . . . .	1
1.3	Acronyms and Definitions . . . . .	1
1.4	Overview of Document . . . . .	1
<b>2</b>	<b>Plan</b>	<b>2</b>
2.1	Software Description . . . . .	2
2.2	Test Team . . . . .	2
2.3	Automated Testing Approach . . . . .	2
2.4	Testing Tools . . . . .	3
2.5	Testing Schedule . . . . .	3
<b>3</b>	<b>System Test Description</b>	<b>3</b>
3.1	Tests for Functional Requirements . . . . .	3
3.1.1	Search Queries . . . . .	3
3.1.2	Download Images . . . . .	12
3.2	Tests for Nonfunctional Requirements . . . . .	14
3.2.1	Look and Feel Requirements . . . . .	14
3.2.2	Usability and Humanity Requirements . . . . .	15
3.2.3	Performance Requirements . . . . .	16
3.2.4	Operational and Environmental Requirements . . . . .	16
3.2.5	Maintainability and Support Requirements . . . . .	17
3.2.6	Legal Requirements . . . . .	17
3.3	Traceability Between Test Cases and Requirements . . . . .	19
<b>4</b>	<b>Tests for Proof of Concept</b>	<b>20</b>
<b>5</b>	<b>Unit Testing Plan</b>	<b>21</b>
5.1	Unit testing of internal functions . . . . .	21
5.2	Unit testing of output files . . . . .	21
<b>6</b>	<b>Appendix</b>	<b>22</b>
6.1	Symbolic Parameters . . . . .	22
6.2	Usability Survey Questions . . . . .	22

## List of Tables

1	<b>Revision History</b>	ii
2	<b>Table of Acronyms</b>	1
3	<b>Table of Definitions</b>	1
4	<b>Functional Requirements Traceability Matrix</b>	19
5	<b>Nonfunctional Requirements Traceability Matrix</b>	19

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	Josh, Sam	2.0	Rev1 Changes: grammar, spelling, and formatting, adding additional functional requirements, providing greater clarity for certain sections, got rid of first person, explicitly defined inputs and outputs for all tests.

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 **gGoogle iImages** downloader command line tool that will allow end users to download a certain number of **images from gGoogle iImages** 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 **gGoogle iImages** query. The second system will be responsible for downloading the images from **gGoogle** 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. **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 Test Cases for Functional Requirements

### 3.1.1 Search Queries

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

**Initial State:** ~~An empty download repository~~ Linux machine has an empty Images folder at google-images-downloader/src/Images

**Input:** ~~A search request with keyword "McMaster"~~ The command `python3 Main.py -k "McMaster"` is provided through the command line

**Output:** ~~Images downloaded all related to McMaster~~ There exists a folder at google-images-downloader/src/Images/McMaster that contains 10 images related to McMaster within 15 seconds.

**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.~~ The tester will use the command line to follow the input steps above and check to ensure the output is correct.

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

**Initial State:** ~~An empty download repository~~ Linux machine has an empty Images folder at google-images-downloader/src/Images

**Input:** ~~A search request with only a specific file type requested~~ The below json object which specifies the gif file type

```
1 {  
2     "keyword": "default",  
3     "limit": 3,  
4     "safesearch": True,  
5     "directory": path.join(curdir, "Images"),  
6     "filetype": "gif",  
7     "colour": "",  
8     "license": "",  
9     "imagetype": "",  
10    "imageage": "",  
11    "aspectratio": "",  
12    "imagesize": "",  
13    "serverhost": "",  
14    "serverusername": "",  
15    "serverpassword": "",
```

```

16     "whitelist": "",
17     "blacklist": "",
18     "region": ""
19 }

```

**Output:** ~~A download repository with only files of the specified type downloaded~~ A folder exists at `google-images-downloader/src/Images/default` that contains three images with the file extension `.gif`

**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 correct.

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

**Initial State:** Linux machine has an empty Images folder at `google-images-downloader/src/Images`

**Input:** The below json object which specifies the jpg file type

```

1 {
2     "keyword": "default",
3     "limit": 3,
4     "safesearch": True,
5     "directory": path.join(curdir, "Images"),
6     "filetype": "jpg",
7     "colour": "",
8     "license": "",
9     "imagetype": "",
10    "imageage": "",
11    "aspectratio": "",
12    "imagesize": "",
13    "serverhost": "",
14    "serverusername": "",
15    "serverpassword": "",
16    "whitelist": "",
17    "blacklist": "",
18    "region": ""
19 }

```



**Output:** A folder exists at google-images-downloader/src/Images/default that contains three images with the file extension .jpg

**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 correct.

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

**Initial State:** Linux machine has an empty Images folder at google-images-downloader/src/Images

**Input:** The below json object which specifies the png file type

```
1 {  
2     "keyword": "default",  
3     "limit": 3,  
4     "safesearch": True,  
5     "directory": path.join(curdir, "Images"),  
6     "filetype": "png",  
7     "colour": "",  
8     "license": "",  
9     "imagetype": "",  
10    "imageage": "",  
11    "aspectratio": "",  
12    "imagesize": "",  
13    "serverhost": "",  
14    "serverusername": "",  
15    "serverpassword": "",  
16    "whitelist": "",  
17    "blacklist": "",  
18    "region": ""  
19 }
```

**Output:** A folder exists at google-images-downloader/src/Images/default that contains three images with the file extension .png

**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 correct.

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

**Initial State:** ~~An empty download repository~~ Linux machine has an empty Images folder at google-images-downloader/src/Images

**Input:** ~~A search query with a request to not download from a specified website~~ The json object below which specifies the blacklist as "mcmaster.ca"

```
1 {
2     "keyword": "mcmaster engineering",
3     "limit": 3,
4     "safesearch": True,
5     "directory": path.join(curdir, "Images"),
6     "filetype": "",
7     "colour": "",
8     "license": "",
9     "imagetype": "",
10    "imageage": "",
11    "aspectratio": "",
12    "imagesize": "",
13    "serverhost": "",
14    "serverusername": "",
15    "serverpassword": "",
16    "whitelist": "",
17    "blacklist": "mcmaster.ca",
18    "region": ""
19 }
```

**Output:** ~~A download repository full with images, with no images from the blacklisted website~~ A list of URLs, none of which contain "mcmaster.ca" in the URL

**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.~~ An automated test case will send the request to the system and check that the list of URLs to be downloaded do not contain the blacklisted site.

FR-SQ6: **Type:** Functional, Dynamic, ~~Manual~~ Automated

**Initial State:** ~~An empty download repository~~ Linux machine has an empty Images folder at google-images-downloader/src/Images

**Input:** ~~A search query with a request to only download from a specified website~~ The json object below which specifies the whitelist as "mcmaster.ca"

```
1 {  
2     "keyword": "engineering",  
3     "limit": 3,  
4     "safesearch": True,  
5     "directory": path.join(curdir, "Images"),  
6     "filetype": "",  
7     "colour": "",  
8     "license": "",  
9     "imagetype": "",  
10    "imageage": "",  
11    "aspectratio": "",  
12    "imagesize": "",  
13    "serverhost": "",  
14    "serverusername": "",  
15    "serverpassword": "",  
16    "whitelist": "mcmaster.ca",  
17    "blacklist": "",  
18    "region": ""  
19 }
```

**Output:** ~~A download repository full with images, with no images from the whitelisted website~~ A list of URLs, all of which contain "mcmaster.ca" in the URL

**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.~~ An automated test case will send the request to the system and check that the list of URLs to be downloaded only contain the whitelisted site.

FR-SQ7: **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-SQ8: **Type:** Functional, Dynamic, Automatic Automated

**Initial State:** An empty download repository Linux machine has an empty Images folder at google-images-downloader/src/Images

**Input:** A search request with the size of desired images specified  
The below json object which specifies a specific image size

```
1 {  
2     "keyword": "homestuck",  
3     "limit": 5,  
4     "safesearch": True,  
5     "directory": path.join(curdir, "Images"),  
6     "filetype": "jpg",  
7     "colour": "",  
8     "license": "",  
9     "imagetype": "",  
10    "imageage": "",  
11    "aspectratio": "",  
12    "imagesize": ">2MP",  
13    "serverhost": "",  
14    "serverusername": "",  
15    "serverpassword": "",  
16    "whitelist": "",  
17    "blacklist": "",  
18    "region": ""  
19 }
```

**Output:** ~~A download repository filled with images of the specified size~~ A folder exists at google-images-downloader/src/Images/homestuck that contains 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.~~ An automated test will send a request to the system, and after all images are downloaded, google-images-downloader/src/Images/homestuck will be iterated through to check that each file matches the specified size.

FR-SQ9: **Type:** Functional, Dynamic, ~~Manual~~ Automated

**Initial State:** ~~An empty download repository~~ Linux machine has an empty Images folder at google-images-downloader/src/Images

**Input:** ~~A search request with all constraints and keywords in a configuration file provided to the system when the user makes the request~~ The below json object and a second json object which is stored at google-images-downloader/src/Test/ConfigForTesting/homestuckFile.txt

```
1 {
2     "keyword": "homestuck",
3     "limit": 5,
4     "safesearch": True,
5     "directory": "",
6     "filetype": "gif",
7     "colour": "green",
8     "license": "",
9     "imagetype": "",
10    "imageage": "",
11    "aspectratio": "",
12    "imagesize": "",
13    "serverhost": "",
14    "serverusername": "",
15    "serverpassword": "",
16    "whitelist": "",
17    "blacklist": "",
18    "region": "Canada"
19 }
```

---

**Output:** ~~Download repository filled with images~~ Two lists of URLs, one built from the above json object, and the one built from the homestuckFile.txt file

**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.~~ Automated test which will build two lists of URLs, one from the json object above, and the other from the textfile, then will compare the two lists to ensure that they are equal.

FR-SQ10: **Type:** Functional, Dynamic, Manual

**Initial State:** Linux machine has an empty Images folder at google-images-downloader/src/Images

**Input:** The command `python3 Main.py -k "fruit" -l 3 -c "red"` is provided through the command line

**Output:** There exists a folder at google-images-downloader/src/Images/fruit that contains 3 images which are red in color

**How test will be performed:** The tester will use the command line to follow the input steps above and check to ensure the output is correct.

FR-SQ11: **Type:** Functional, Dynamic, Manual

**Initial State:** Linux machine has an empty Images folder at google-images-downloader/src/Images

**Input:** The command `python3 Main.py -k "waterfall" -l 3 -ar "panoramic"` is provided through the command line

**Output:** There exists a folder at google-images-downloader/src/Images/waterfall that contains 3 images which are panoramas

**How test will be performed:** The tester will use the command line to follow the input steps above and check to ensure the output is correct.

### 3.1.2 Download Images

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

**Initial State:** ~~An empty download repository~~ Linux machine has an empty Images folder at google-images-downloader/src/Images

**Input:** ~~A search request with a limit of 50 images specified~~ The below json object which specifies a limit of 40 images

```
1 {  
2     "keyword": "software",  
3     "limit": 40,  
4     "safesearch": True,  
5     "directory": path.join(curdir, "Images"),  
6     "filetype": "",  
7     "colour": "",  
8     "license": "",  
9     "imagetype": "",  
10    "imageage": "",  
11    "aspectratio": "",  
12    "imagesize": "",  
13    "serverhost": "",  
14    "serverusername": "",  
15    "serverpassword": "",  
16    "whitelist": "",  
17    "blacklist": "",  
18    "region": ""  
19 }
```

**Output:** ~~A download repository contains 50 images in it~~ A folder exists at google-images-downloader/src/Images/software that contains 40 images

**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.~~ An automated test will send a request to the system, and after all images are downloaded, google-images-downloader/src/Images/software will be iterated through to check that there are 40 images.

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

**Initial State:** ~~An empty download repository~~ Linux machine has an empty Images folder at google-images-downloader/src/Images

**Input:** ~~A download request with a specified download folder~~ The below json object which specifies a download folder

```
1 {  
2     "keyword": "software",  
3     "limit": 2,  
4     "safesearch": True,  
5     "directory": path.join(curdir, "newDirec"),  
6     "filetype": "",  
7     "colour": "",  
8     "license": "",  
9     "imagetype": "",  
10    "imageage": "",  
11    "aspectratio": "",  
12    "imagesize": "",  
13    "serverhost": "",  
14    "serverusername": "",  
15    "serverpassword": "",  
16    "whitelist": "",  
17    "blacklist": "",  
18    "region": ""  
19 }
```

**Output:** ~~Images downloaded to the specified folder~~ A folder exists at google-images-downloader/src/newDirec/software that contains 2 images

**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.~~ An automated test will send a request to the system, and after all images are downloaded, google-images-downloader/src/newDirec/software will be iterated through to check that there are 2 images.



FR-DL3: **Type:** Functional, Dynamic, Manual

**Initial State:** Linux machine has an empty Images folder at google-images-downloader/src/Images

**Input:** The command `python3 Main.py -k "McMaster" -l 3 -s "moore.mcmaster.ca" -u "guinnesj" -p "password"` is provided through the command line

**Output:** There exists a folder at `moore.mcmaster.ca:/u50/guinnesj/McMaster` that contains 3 images within 15 seconds

**How test will be performed:** The tester will use the command line to follow the input steps above and check to ensure the output is correct by logging into the server.

## 3.2 Test Cases 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~~ Program is not running.

**Input/Condition:** ~~A typical usage of the system~~ The command `python3 Main.py -k "waterfall" -l 3` is provided through the command line

**Output/Result:** Output messages displayed on the console are fully visible

**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.~~ The tester will use the command line to follow the input step above and check to ensure the output is correct.

### 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~~ Program is not running.

**Input/Condition:** ~~A typical usage of the system~~ The command `python3 Main.py -k "waterfall" -l 3` is provided through the command line

**Output/Result:** ~~A folder of the desired downloaded images~~ A folder exists at `google-images-downloader/src/Images/waterfall` which has three 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~~ The tester will ensure that the sample inputs and explanations are displayed upon a faulty execution, and that a correct execution provides the specified download to occur.

#### 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~~ A series of commands entered through the command line:

```
$ git clone https://gitlab.cas.mcmaster.ca/guinnessj  
  ↪ /google-images-downloader  
$ cd google-images-downloader/src  
$ make install
```

**Output/Result:** ~~A folder of the desired downloaded images~~ Program exists on local system and dependencies are installed.

**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.~~ The tester will ensure that the program and all required libraries can easily be downloaded by following the above steps.

### 3.2.3 Performance Requirements

#### Robustness Requirements

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

**Initial State:** ~~A clean version of the program~~ Program is not running.

**Input/Condition:** ~~An invalid input; an input that would result in an error~~ The command `python3 Main.py` is provided to the command line.

**Output/Result:** A descriptive error message showing the possible command line arguments is shown

**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.~~ The tester will run the input steps above and confirm the output is true.

### 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~~ Most recently updated system

**Input/Condition:** ~~A typical usage of the program~~ The command `python3 Main.py -k mcmaster` is provided to the command line

**Output/Result:** ~~A folder of the desired downloaded images~~ A folder exists at google-images-downloader/src/images with 10 images.

**How test will be performed:** ~~The tester will manually verify that the program can download all the images using the latest format.~~ The tester will run the input steps above and confirm the output is true.

### 3.2.5 Maintainability and Support Requirements

#### Adaptability Requirements

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

**Initial State:** The most up to date version of the program on a Linux, Windows, and Mac platform.

**Input/Condition:** ~~A typical usage of the program~~ The command `python3 Main.py -k mcmaster` is provided to the command line

**Output/Result:** ~~A folder of the desired downloaded images~~ A folder exists at google-images-downloader/src/images with 10 images.

**How test will be performed:** The tester will manually verify that ~~all test cases pass on all platforms~~ given the input, the output is true on all platforms

### 3.2.6 Legal Requirements

#### Standards Requirements

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

**Initial State:** ~~A clean version of the program~~ The most up to date version of the program on a Linux, Windows, and Mac platform.

**Input/Condition:** ~~An execution of the linter~~ The command `make lint` is provided to the command line under the `src/` directory

**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	FR10	FR11	FR12	FR13	FR14	FR15
FR-SQ1	✓														
FR-SQ2				✓											
FR-SQ3				✓											
FR-SQ4				✓											
FR-SQ5					✓										
FR-SQ6						✓									
FR-SQ7															
FR-SQ8								✓							
FR-SQ9									✓						
FR-SQ10							✓								
FR-SQ11											✓				
FR-DL1		✓													
FR-DL2			✓												
FR-DL3										✓					

Table 4: Functional Requirements Traceability 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 Traceability Matrix

The following Nonfunctional Requirements are not included in the Traceability 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.
- HSR1: Protection of our eyes.

## 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 results page of Google Images.** 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 that will be used are 85% 80% statement coverage and 80% branch coverage. This testing will not include the Main module, as it just facilitates the execution of the other modules.~~ 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.~~ Since all testing will be performed by the development team, there are no survey questions that will be used to test the software. However, any input provided via the git repository will be taken into account by the developers of the program.