SE 3XA3: Module Guide
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

Date	Name	Version	Notes
2/26/2020	Nick	1.0	Created document
3/9/2020	Sam	1.1	Filled in admin details
3/9/2020	Sam	1.1.1	Started adding modules
3/10/2020	Josh, Sam, Nick	1.1.2	First iteration of all sections completed
3/11/2020	Sam, Nick	1.2	Finishing touches
3/13/2020	Josh	1.2.1	Finishing section 2, general edits
3/13/2020	Nick	1.2.2	General edits
4/6/2020	Sam	2.0	Final revision for Rev 1

Table 1: **Revision History**

1 Introduction

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

After completing the first stage of the design, the Software Requirements Specification (SRS), this document, the Module Guide (MG) is developed (Parnas et al., 1984). The MG specifies the modular structure of the system and is intended to allow both designers and maintainers to easily identify the parts of the software. After the MG is completed, the Module Interface Specification will be done which specifies the externally observable behaviour of a module using precise mathematical language.

A module is a work assignment for a programmer or programming team (Parnas et al., 1984). The primary design principle being followed when decomposing the system into modules is the principle of information hiding (Parnas, 1972). This principle supports design for change, because the "secrets" that each module hides represent likely future changes.

Other design principles follow the rules laid out by Parnas et al. (1984). They are as follows:

- System details that are likely to change independently should be the secrets of separate modules.
- Each data structure is used in only one module.
- Any other program that requires information stored in a module's data structures must obtain it by calling access programs belonging to that module.

The rest of the document is organized as follows. Section 2 lists the anticipated and unlikely changes of the software requirements. Section 3 summarizes the module decomposition that was constructed according to the likely changes. Section 4 specifies the connections between the software requirements and the modules. Section 5 gives a detailed description

of the modules. Section 6 includes two traceability matrices. One checks the completeness of the design against the requirements provided in the SRS. The other shows the relation between anticipated changes and the modules. Section 7 describes the use relation between modules.

2 Anticipated and Unlikely Changes

This section lists possible changes to the system that are classified into two categories according to the likeliness of the change. Anticipated changes are listed in Section 2.1, and unlikely changes are listed in Section 2.2. Table 4 shows the traceability matrix between the anticipated changes and modules.

2.1 Anticipated Changes

Anticipated changes are the source of the information that are to be hidden inside the modules. This was, changing one of the anticipated changes will only require changing the one module that hides the associated decision. The is called design for change.

AC1: Google Images will likely change the way that their HTML is structured in the future. This will require a change in the Navigate Page Module.

AC2: Google Images will likely change the search query format, requiring a change in the Search Query Module.

AC3: Google Images may change the search paramaters available to use, resulting in a change in the Input Module.

2.2 Unlikely Changes

These are changes that are unlikely to happen as changing them will likely require many parts of the design to be modified. This is obviously not ideal.

UC1: Input/Output devices (Input: File and/or Keyboard, Output: Images downloaded to a directory).

UC2: The system will take in user input from either from the command line or from a file.

UC3: The system will download images to either a local folder, or a specified location on a server.

UC4: The program will run on a computer or server that has the necessary tools downloaded and installed, not an embedded system.

UC5: The data flow and ~~logic of the system~~ system architecture is unlikely to change.

3 Module Hierarchy

This section provides an overview of the module design. Modules are summarized in a hierarchy decomposed by secrets in Table 2. The modules listed below are leaves in the hierarchy tree and will actually be implemented.

M1: Hardware-Hiding Module

M2: Input Format Module

M3: Search Query Module

M4: Navigate Page Module

M5: Output Format Module

M6: Main Module

Level 1	**Level 2**
Hardware-Hiding Module	
Behaviour-Hiding Module	Input Module Search Query Module Navigate Page Module Output Module ~~Main Module~~
Software Decision Module	Main Module

Table 2: Module Hierarchy

4 Connection Between Requirements and Design

The design of the system is intended to satisfy the requirements developed in the SRS. In this stage, the system is decomposed into modules with high cohesion and low coupling, to implement proper software design principles. These include the connection between requirements and modules is listed in Table 3.

5 Module Decomposition

Modules are decomposed according to the principle of "information hiding" proposed by Parnas et al. (1984). The *Secrets* field in a module decomposition is a brief statement of the design decision hidden by the module. The *Services* field specifies *what* the module will do without documenting *how* to do it. For each module, a suggestion for the implementing software is given under the *Implemented By* title. If the entry is *OS*, this means that the module is provided by the operating system or by standard programming language libraries. If the entry is *Google Images Downloader*, this means that the module will be implemented specifically for the software.

Only the leaf modules in the hierarchy have to be implemented. If a dash (–) is shown, this means that the module is not a leaf and will not have to be implemented. Whether or not this module is implemented depends on the programming language selected.

5.1 Hardware Hiding Modules (M1)

Secrets: The data structure and algorithm used to implement the virtual hardware.

Services: Serves as a virtual hardware used by the rest of the system. This module provides the interface between the hardware and the software. So, the system can use it to display outputs or to accept inputs.

Implemented By: OS

5.2 Behaviour-Hiding Module

Secrets: The contents of the required behaviours.

Services: Includes programs that provide externally visible behaviour of the system as specified in the software requirements specification (SRS) documents. This module serves as a communication layer between the hardware-hiding module and the software decision module. The programs in this module will need to change if there are changes in the SRS.

Implemented By: –

5.2.1 Input Format Module (M2)

Secrets: The format and structure of the input data.

Services: Converts the input data into the data structure used by the Search Query Module.

Implemented By: Google Images Downloader

5.2.2 Search Query Module (M3)

Secrets: The format and structure of the Google search query.

Services: Converts the processed data from the Input Module into a URL string representing the desired Google search query.

Implemented By: Google Images Downloader

5.2.3 Navigate Page Module (M4)

Secrets: The layout of the Google search page.

Services: Uses the processed Google search query to obtain the URL of each individual image.

Implemented By: Google Images Downloader

5.2.4 Output Format Module (M5)

Secrets: The format and structure of the output data.

Services: Converts the processed data into output files/folders that will be written to storage.

Implemented By: Google Images Downloader

5.3 Software Decision Module

Secrets: The design decision based on mathematical theorems, physical facts, or programming considerations. The secrets of this module are *not* described in the SRS.

Services: Includes data structure and algorithms used in the system that do not provide direct interaction with the user.

Implemented By: –

5.3.1 Main Module (M6)

Secrets: The control flow logic of the system.

Services: Controls the flow of data.

Implemented By: Google Images Downloader

6 Traceability Matrix

This section shows two traceability matrices: between the modules and the requirements and between the modules and the anticipated changes.

Req.	Modules
FR1	M2, M3, M5
FR2	M2, M4
FR3	M2, M5
FR4	M2, M3
FR5	M2, M4
FR6	M2, M3
FR7	M2, M3
FR8	M2, M3
FR9	M1, M2, M3, M4, M5
FR10	M2, M5
FR11	M2, M3
FR12	M2, M3
FR13	M2, M3
FR14	M2, M3
FR15	M2, M3
EUR1	M1, M5
LR1	–
LR2	–
RR1	M2
IAR1	M3, M4
PDR1	–
MSR1	–
MSR2	–
ADR1	M5
ACR1	–
CR1	M2
CPR1	–
SR1	M2, M3, M4, M5, M6
HSR1	M2, M3

Table 3: Trace Between Requirements and Modules

AC	Modules
AC1	M4
AC2	M3
AC3	M2

Table 4: Trace Between Anticipated Changes and Modules

7 Use Hierarchy Between Modules

In this section, the uses hierarchy between modules is provided. Parnas (1978) said of two programs A and B that A *uses* B if correct execution of B may be necessary for A to complete the task described in its specification. That is, A *uses* B if there exist situations in which the correct functioning of A depends upon the availability of a correct implementation of B. Figure 1 illustrates the use relation between the modules. It can be seen that the graph is a directed acyclic graph (DAG). Each level of the hierarchy offers a testable and usable subset of the system, and modules in the higher level of the hierarchy are essentially simpler because they use modules from the lower levels.

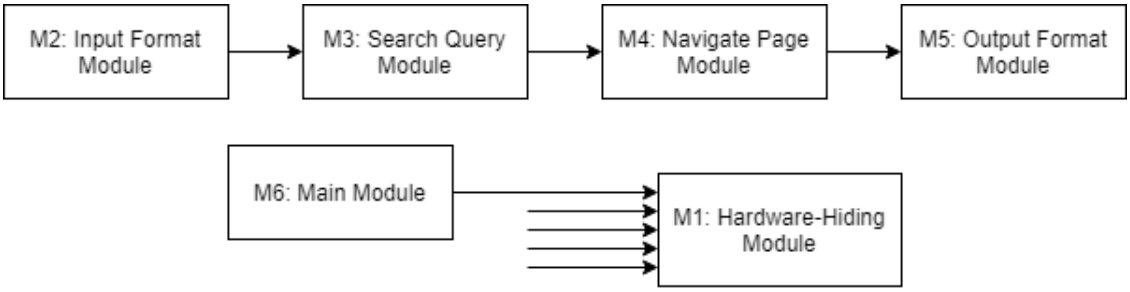

Figure 1: Use hierarchy among modules

References

David L. Parnas. On the criteria to be used in decomposing systems into modules. *Comm. ACM*, 15(2):1053–1058, December 1972.

David L. Parnas. Designing software for ease of extension and contraction. In *ICSE '78: Proceedings of the 3rd international conference on Software engineering*, pages 264–277, Piscataway, NJ, USA, 1978. IEEE Press. ISBN none.

D.L. Parnas, P.C. Clement, and D. M. Weiss. The modular structure of complex systems. In *International Conference on Software Engineering*, pages 408–419, 1984.