

SE 3XA3: Module Guide

Gifitti

Team #2, Gifitti
Pavle Arezina, arezinp
Nicolai Kozel, kozeln
Riley McGee, mcgeer

November 30, 2016

Contents

1	Introduction	1
2	Anticipated and Unlikely Changes	2
2.1	Anticipated Changes	2
2.2	Unlikely Changes	3
3	Module Hierarchy	3
4	Connection Between Requirements and Design	4
5	Module Decomposition	6
5.1	Hardware Hiding Modules (M1)	6
5.2	Behaviour-Hiding Module (M2)	6
5.2.1	Image Processing Module (M4)	7
5.2.2	Image Loading Module (M5)	7
5.2.3	GIF Model Module (M8)	7
5.2.4	View-Model Module (M9)	7
5.3	Software Decision Module (M3)	8
5.3.1	GIF Transformation Module (M7)	8
5.3.2	Image Conversion Module (M6)	9
6	Traceability Matrix	9
7	Use Hierarchy Between Modules	10

List of Tables

1	Revision History	ii
2	Module Hierarchy	4
3	Trace Between Requirements and Modules	9
4	Trace Between Anticipated Changes and Modules	10

List of Figures

1	Use hierarchy among modules	11
---	---------------------------------------	----

Table 1: **Revision History**

Date	Version	Notes
November 9th 2016	1.0	Added Module Hierarchy
November 9th 2016	1.1	Added Module Decomposition
November 11th 2016	1.2	Added Section 2 and 6
November 13th 2016	2.0	Spelling and grammatical issues ad- dressed

1 Introduction

Decomposing a system into modules is a commonly accepted approach to developing software. A module is a work assignment for a programmer or programming team (Parnas et al., 1984). We advocate a decomposition based on the principle of information hiding (Parnas, 1972). This principle supports design for change, because the “secrets” that each module hides represent likely future changes. Design for change is valuable in SC, where modifications are frequent, especially during initial development as the solution space is explored.

Our design follows the rules laid out by Parnas et al. (1984), 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.

Gifitti is a software solution for allowing users with a varying knowledge base on image manipulation, as well as technical experience with computers to be able to load, decompose, and modify GIF images.

The following document outlines the Module Guide for Gifitti adhering to the document description above.

After completing the first stage of the design, the Software Requirements Specification (SRS), 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. The potential readers of this document are as follows:

- New project members: This document can be a guide for a new project member to easily understand the overall structure and quickly find the relevant modules they are searching for.
- Maintainers: The hierarchical structure of the module guide improves the maintainers’ understanding when they need to make changes to the system. It is important for a maintainer to update the relevant sections of the document after changes have been made.

- Designers: Once the module guide has been written, it can be used to check for consistency, feasibility and flexibility. Designers can verify the system in various ways, such as consistency among modules, feasibility of the decomposition, and flexibility of the design.

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. According to the likeliness of the change, the possible changes are classified into two categories. Anticipated changes are listed in Section 2.1, and unlikely changes are listed in Section 2.2.

2.1 Anticipated Changes

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

AC1: The specific hardware on which the software is running.

AC2: The types of available output files.

AC3: The type of operating systems the program runs on.

AC4: Added functions to the GIF editor (ex. Changing colour to all frames)

AC5: The graphical user interface the user utilizes to access program.

2.2 Unlikely Changes

The module design should be as general as possible. However, a general system is more complex. Sometimes this complexity is not necessary. Fixing some design decisions at the system architecture stage can simplify the software design. If these decision should later need to be changed, then many parts of the design will potentially need to be modified. Hence, it is not intended that these decisions will be changed.

UC1: Input/Output devices (Input: File and/or Keyboard, Output: File, Memory, and/or Screen).

UC2: There will always be a source of input data external to the software.

UC3: The input file being a GIF file.

UC4: The algorithm utilized to select and edit frames.

UC5: The purpose of the program: To edit GIFs.

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, which are leaves in the hierarchy tree, are the modules that will actually be implemented.

M1: Hardware-Hiding Module

M2: Behaviour-Hiding Module

M3: Software Decision Module

M4: Image Processing Module

M5: Image Loading Module

M6: Image Conversion Module

M7: GIF Transformation Module

M8: GIF Model Module

M9: View-Model Module

The Hardware-Hiding Module is Handled in full by the C Sharp Sytem Libraries, and requires no further implementation.

Level 1	Level 2
Hardware-Hiding Module	
Behaviour-Hiding Module	View-Model Module GIF Model Module Image Loading Module Image Processing Module
Software Decision Module	GIF Transformation Module Image Conversion 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. The connection between requirements and modules is listed in Table 3. Below is a simple description of the design decisions and how they impact the major functional and non functional requirements.

1. User can provide input and get output from the system (i.e open and save a GIF).

This requirement is satisfied through the use of buttons/file dialogs and is handled primary through C#'s built in libraries as well as the Image Loading Module and Image Conversion Module.

2. Once a GIF is loaded by the user, it must playback in the window and playback must be able to be stopped and started by the user.

This requirement is primarily handled by the GIF Model Module. To ensure this requirement is met, the design is being implemented so that the GIF Model will handle the frames and functions of playing back the GIF.

3. The user must be able to modify the GIF including clipping, frame injection, and possibly add text or other make modifications.

This requirement is met through the GIF Transformation Module. This module contains all the possibilities of modifications that can be applied to the GIF. These functions are separated from the GIF Model Module to provide us with the ability to make changes to the model without affecting the behavior of the system and having to modify all of the transformation functions. These functions will also be implemented through the use of buttons and click gestures.

4. In general, the system must operate at a high performance rate. This includes a high frame rate during playback and fast/responsive UI elements and load times.

Some design decisions that were made to ensure this were how the modules were decomposed and the choice of image framework to work with. The system is decomposed in such a way to ensure the code is efficient and any algorithms used operate at an optimal rate. The UI elements and file manipulation methods were specifically chosen to be the fastest possible to ensure a positive experience for the user.

5. The program should have a clean UI design and easily usable by people older than 10 years old.

Design decisions were made during the implementation of the system to ensure this requirement. These included the size of the buttons, the font size/colors, and the general layout. The original application (GIF Viewer) had a fairly usable interface already, so we simply expanded off of this but attempted to make it even simpler and cleaner by eliminating unnecessary and confusing submenus. Instead, every button serves

a single purpose.

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. Also indicate if 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 (M2)

Secrets: The contents of the required software system 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 Image Processing Module (M4)

Secrets: The format and structure of GIF, TIFF, JPEG, PNG, and BMP image types, and Image Meta data parsing is handled by this module.

Services: Converts the input images into a system usable form.

Implemented By: ImageMagick.NET

5.2.2 Image Loading Module (M5)

Secrets: Utilizes the Image Processing Module to read in image data and return a system useable form.

Services: Converts system paths of images to a useable form in the software.

Implemented By: Gifitti GIF Model Module (M8)

5.2.3 GIF Model Module (M8)

Secrets: GIF images are represented in a Frame-By-Frame manner to establish easy handling of GIFs. The images themselves are represented as Bitmaps to keep

Services:

- Hold initial GIF image for data security.
- Make GIF meta data available and editable.
- Represent how the user will view the GIF in the system.

Implemented By: Gifitti

5.2.4 View-Model Module (M9)

Secrets: Views are connected to View-Models via C# partial class declarations. View-Models are a C# architecture, to allow Views to connect to their Model representation with a loose coupling. This allows for Views and their Models to be developed independently from each other, in a

parallel fashion. The partial class connection of the View-Model and View elements is what keeps Views and View-Models in the same Module, although they are abstracted from each other.

Services: Controls View elements, as well as system interactions by the user.

Services:

- Controls View elements, as well as system interactions by the user (View-Models).
- Essential link for View elements and Models

Implemented By: Gifitti

5.3 Software Decision Module (M3)

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 GIF Transformation Module (M7)

Secrets: Frame-By-Frame Manipulation of GIF Models to obtain modified GIFs.

Services: Converts GIFs into a new GIF based on desired transformations. Transformations include: Reset, Resize, Add Frame(s), Remove Frame(s), Rotate, Modify Speed ...

Implemented By: Gifitti

5.3.2 Image Conversion Module (M6)

Secrets: Converts GIF Models into specific Image types, using ImageMagick.NET

Services: Allows a GIF in the system to be exported as a desired image type. Image types include GIF, TIFF, JPEG, PNG, and BMP.

Implemented By: Gifitti GIF Model Module, ImageMagick.Net (M8)

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
R1	M1, M2, M3, M5
R2	M1, M2, M3, M5
R3	M1, M2, M3, M5
R4	M3, M5
R5	M1, M2, M3, M5, M6, M8
R6	M9, M8
R7	M9, M8
R8	M9, M8
R9	M7, M9, M8
R10	M9, M8, M6
R11	M4, M5
R12	M4
R13	M8, M9
R14	M8, M9
R15	M9
R16	M7
R17	M7
R18	M7, M9
R19	M7, M9
R20	M7, M9

Table 3: Trace Between Requirements and Modules

AC	Modules
AC1	M1
AC2	M5
AC3	M2
AC4	M7
AC5	M9

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.

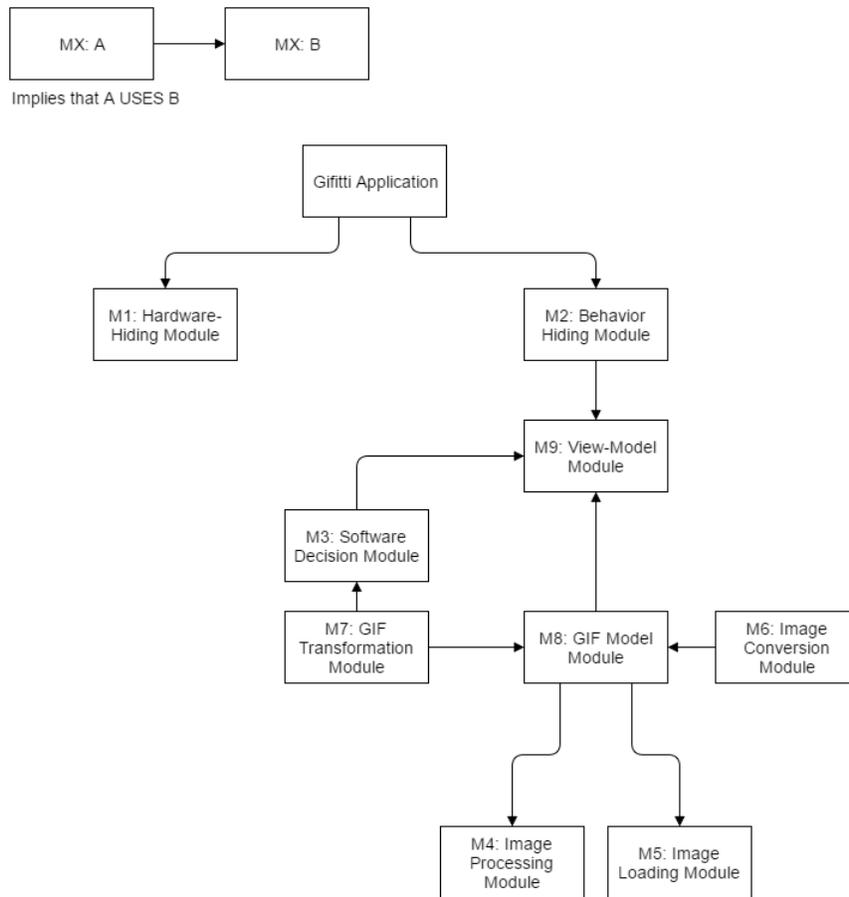


Figure 1: Use hierarchy among modules

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.