Module Guide for PolyHarmonics

Nolan Driessen

August 26, 2015

Contents

1 Introduction **2**

2 Anticipated and Unlikely Changes **3**
- 2.1 Anticipated Changes . . . 3
- 2.2 Unlikely Changes . . . 3

3 Module Hierarchy **4**

4 Connection Between Requirements and Design **4**

5 Module Decomposition **5**
- 5.1 Hardware Hiding Modules (M1) . . . 5
- 5.2 Behaviour-Hiding Module . . . 5
  - 5.2.1 Input Finding Module (M2) . . . 6
  - 5.2.2 Input Data Module (M3) . . . 6
  - 5.2.3 Transformed Signal Data Module (M4) . . . 6
  - 5.2.4 Plot Data Module (M6) . . . 6
  - 5.2.5 Control Module (M7) . . . 6
  - 5.2.6 Output Module (M8) . . . 7
  - 5.2.7 Filtered Data (M10) . . . 7
- 5.3 Software Decision Module . . . 7
  - 5.3.1 Signal Transforming Module (M5) . . . 7
  - 5.3.2 Filtering Module (M9) . . . 7

6 Traceability Matrix **7**

7 Use Hierarchy Between Modules **8**

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 (?). In the best practices for scientific computing, ?) advise a modular design, but are silent on the criteria to use to decompose the software into modules. We advocate a decomposition based on the principle of information hiding (?). 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 ?), 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.

After completing the first stage of the design, the Software Requirements Specification (SRS), the Module Guide (MG) is developed (?). 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 information contained within the output text files.

AC3: The method of finding and using input files.

AC4: The method of storing required information found within input files.

AC5: The way the plots are created and displayed.

AC6: The components of the signal being analyzed.

AC7: The method used for analysis of the acoustic signals.

AC8: The control flow of the system.

AC9: The method of filtering the input data.

AC10: The method of storing the filtered data.

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: Keyboard and/or mouse, Output: Memory and/or Screen).

UC2: Output data is displayed to the output device.

UC3: The goal of the system is to interpret acoustic signals.

UC4: The Discrete Fourier Transform is used.

UC5: The system receives input from a LabVIEW TDMS file as described by section 4.2 in the Software Requirements Specification.

UC6: Plots are created containing critical information regarding the signals analyzed.

UC7: The system gives its output to ProMV after analysis as described by section 4.2 in the Software Requirements Specification. [You could reference the system constraints in the SRS at this point. —SS]

[Both spots referenced —ND]

3 Module Hierarchy

This section provides an overview of the module design. Modules are summarized in a hierarchy decomposed by secrets in Table 1. The modules listed below, which are leaves in the hierarchy tree, are the modules that will actually be implemented.

M1: Hardware-Hiding Module

M2: Input Finding Module

M3: Input Data Module

M4: Transformed Signal Data Module

M5: Signal Transforming Module

M6: Plot Data Module

M7: Control Module

M8: Output Module

M9: Filtering Module

M10: Filtered Data Module

Note that M1 is a commonly used module and is already implemented by the operating system. It will not be reimplemented.

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 2.

Table 1: Module Hierarchy

Level 1	**Level 2**	**Level 3**
Hardware-Hiding Module		
Behaviour-Hiding Module	Input Module	Input Finding Module Input Data Module
	Transformed Signal Data Module Plot Data Module Control Module Output Module Filtered Data Module	
Software Decision Module	Signal Transforming Module Filtering Module	

5 Module Decomposition

Modules are decomposed according to the principle of "information hiding" proposed by **?**). 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 PolyHarmonics, this means that the module will be implemented by the acoustic analysis software. If the entry is *NumPy*, *Matplotlib* or *Pywt* it means the module will be implemented by that specific Python library. 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 behaviors.

Services: Includes programs that provide externally visible behavior 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 Finding Module (M2)

Secrets: Encapsulates finding files for input and accepting user input.

Services: Searches for TDMS files within the given directory and accepts starting stopping and step frequencies from the user.

Implemented By: PolyHarmonics

5.2.2 Input Data Module (M3)

Secrets: The format and structure of the data found within the input files.

Services: Stores the data from the Input Module.

Implemented By: PolyHarmonics

5.2.3 Transformed Signal Data Module (M4)

Secrets: The format and structure of the data of the transformed signal.

Services: Stores the data of the transformed signal.

Implemented By: PolyHarmonics

5.2.4 Plot Data Module (M6)

Secrets: Encapsulates the creation of plots.

Services: Creates plots of the input data and transformed data.

Implemented By: PolyHarmonics

5.2.5 Control Module (M7)

Secrets: The algorithm for coordinating the running of the program.

Services: Provides the main program.

Implemented By: PolyHarmonics

5.2.6 Output Module (M8)

Secrets: The format and structure of the output data.

Services: Writes the output data to the output files in the proper format.

Implemented By: PolyHarmonics

5.2.7 Filtered Data (M10)

Secrets: The format and structure of the filtered data.

Services: Stores the data of the filtered signal.

Implemented By: PolyHarmonics

5.3 Software Decision Module

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

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

Implemented By: –

5.3.1 Signal Transforming Module (M5)

Secrets: Encapsulates taking an acoustic signal in the time domain and transforming it to the frequency domain.

Services: Handles the algorithm for taking the Discrete Fourier Transform.

Implemented By: NumPy

5.3.2 Filtering Module (M9)

Secrets: Encapsulates applying a filter to the input data.

Services: Handles applying the Haar wavelet algorithm.

Implemented By: Pywt

6 Traceability Matrix

This section provides two traceability matrices: between the modules and the requirements, Table 2, and between the modules and the anticipated changes, Table 3.

Table 2: Trace Between Requirements and Modules

Req.	**Modules**
R1	M1, M2, M7
R2	M3, M7
R3	M3, M7, M9, M10
R4	M3, M4, M5, M7, M10
R5	M6, M7
R6	M6, M7
R7	M6, M7
R8	M3, M7, M8
R9	M7, M8, M10
R10	M3, M7, M8

Table 3: Trace Between Anticipated Changes and Modules

A.C.	**Modules**
AC1	M1
AC2	M8
AC3	M2
AC4	M3
AC5	M6
AC6	M4
AC7	M5
AC8	M7
AC9	M9
AC10	M10

7 Use Hierarchy Between Modules

In this section, the uses hierarchy between modules is provided. **?**) 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.

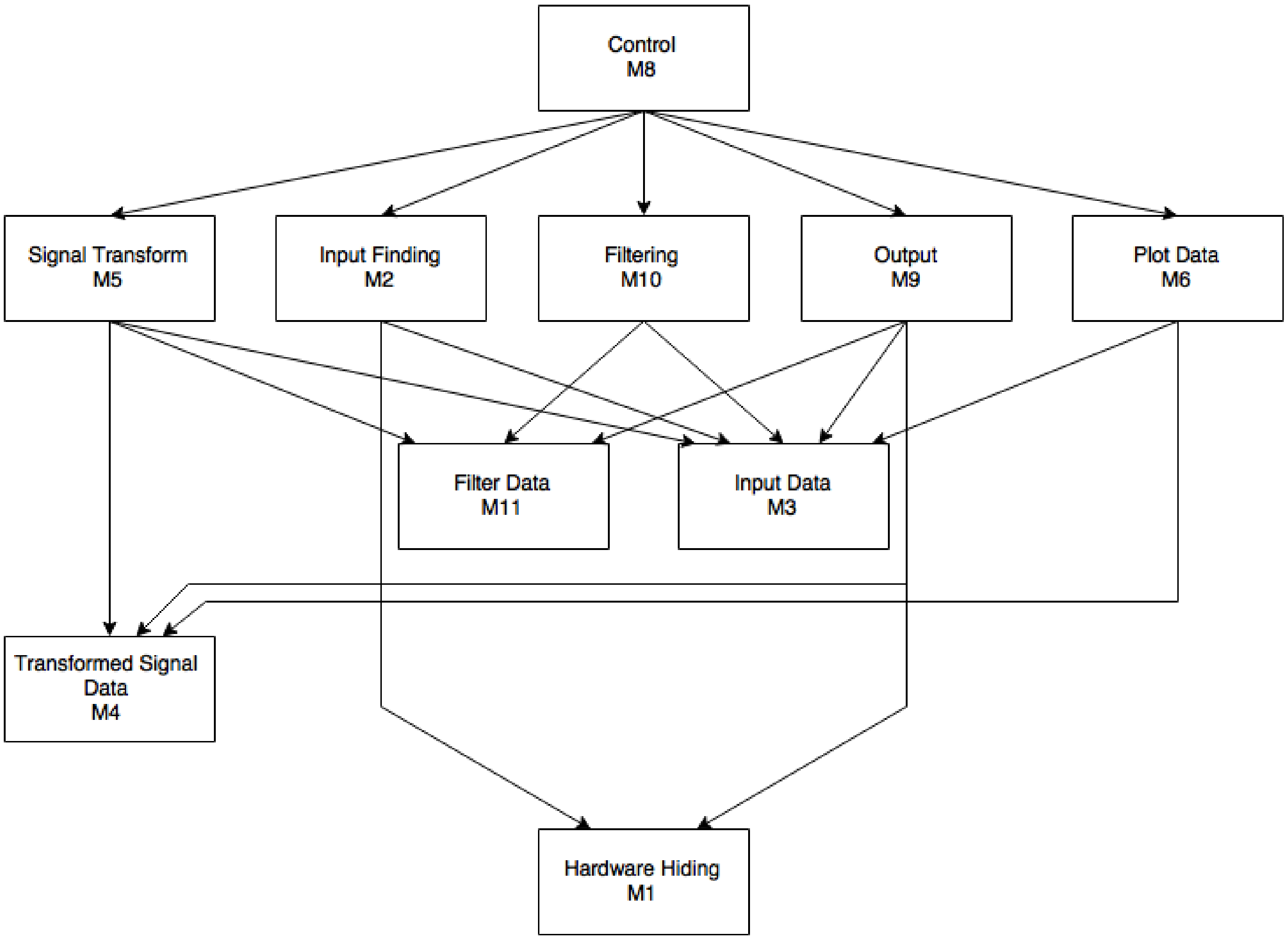

Figure 1: Use hierarchy among Modules