# Module Guide: Curve Fitting Software

Malavika Srinivasan

November 4, 2018

# 1 Revision History

| Date | Version | Notes |
|---|---|---|
| Oct 27, 2018 | 1.0 | First draft by Malavika |
| Oct 30, 2018 | 1.1 | Second draft by Malavika |

# Contents



# List of Tables



# List of Figures

# 2 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 layed 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.

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 3 lists the anticipated and unlikely changes of the software requirements. Section 4 summarizes the module decomposition that was constructed according to the likely changes. Section 5 specifies the connections between the software requirements and the modules. Section 6 gives a detailed description of the modules. Section 7 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 8 describes the use relation between modules.

# 3 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 3.1, and unlikely changes are listed in Section 3.2.

## 3.1 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 format of the initial input data.[May be manual instead of file input, or instead of t arrays and y arrays, they may want to input data as a tuple of t,y —Malavika]

**AC3:** The method used to calculate best fit may be smoothing instead of interpolation and regression.

**AC4:** The format of output data. Instead of best fit coefficients and plot, they may want to obtain the value of 'y' at a particular 't' or a series of 't' values where $(t_i, y_i)$ is the input data at $i = 0, 1, 2, ...n$.

**AC5:** The user may not want to input the degree of the polynomial for regression and my want it to be computed by CFS.

## 3.2 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. [Not sure if this applies to my software —Malavika]

**UC3:** If CFS is used with non linear systems, then the results from the software is no longer valid.

# 4 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:** Behaviour-Hiding Module

**M3:** User-program Module

**M4:** Input Module

**M5:** Best fit Module

**M6:** Output Module

**M7:** Software Decision Module

**M8:** Sequence Services Module

**M9:** Interpolation Module

**M10:** Linear Regression Module

**M11:** Plot Module

Note that M1 is a commonly used module and is already implemented by the operating system. It will not be reimplemented. Similarly, M8 and M11 are already available in Python and will not be reimplemented.

| **Level 1** | **Level 2** |
|---|---|
| Hardware-Hiding Module | |
| Behaviour-Hiding Module | Input<br>Best fit<br>Output |
| Software Decision Module | Sequence Services<br>Interpolation<br>Linear Regression<br>Plot |

Table 1: Module Hierarchy

# 5 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.

# 6 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.

## 6.1 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

## 6.2 Behaviour-Hiding Module (M2)

**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:** –

#### 6.2.1 User-program Module (M3)

**Secrets:** This module uses the best fit module to use the appropriate service.

**Services:** NA.

**Implemented By:** User

#### 6.2.2 Input Module (M4)

**Secrets:** The format and structure of input data.

**Services:** Converts the input data into the data structure used by other modules and verifies the type of the input. A 'TypeError' exception is raised if there is a type mismatch.

**Implemented By:** CFS

#### 6.2.3 Best fit Module (M5)

**Secrets:** The user's choice of best fit method and degree of the fit.

**Services:** Provides the interface to invoke the appropriate method of fit and obtains additional input like degree of the polynomial for regression.

**Implemented By:** CFS

#### 6.2.4 Output Module (M6)

**Secrets:** The format and structure of the output data.

**Services:** Outputs the best fit parameters and plots the output along with input data and best fit parameters.

**Implemented By:** CFS

### 6.3 Software Decision Module (M7)

**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:** –

### 6.3.1 Sequence Services Module (M8)

**Secrets:** The data structure for a sequence data type.

**Services:** Provides array manipulation, including building an array, accessing a specific entry etc.

**Implemented By:** Python

### 6.3.2 Interpolation Module (M9)

**Secrets:** The algorithms used for interpolation

**Services:** Provides the best fit coefficients

**Implemented By:** CFS

### 6.3.3 Linear Regression (M10)

**Secrets:** The algorithm used for linear regression

**Services:** Provides the best fit coefficients

**Implemented By:** CFS

### 6.3.4 Plot Module (M11)

**Secrets:** The data structures and algorithms for plotting data graphically.

**Services:** Provides a plot function.

**Implemented By:** Python

# 7 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, M4, M8 |
| R2 | M2, M3 M4, M8 |
| R3 | M2, M3 M4, M8 |
| R4 | M1,M2, M3, M5 |
| R5 | M1,M2, M3, M5 |
| R6 | M1,M2, M3, M5 |
| R7 | M1,M2, M3, M5 |
| R8 | M1,M2, M3, M5 |
| R9 | M3,M7, M8, M9, M10 |
| R10 | M1,M2, M6, M8, M11 |

Table 2: Trace Between Requirements and Modules

| AC | Modules |
|---|---|
| AC1 | M1 |
| AC2 | M4 |
| AC3 | M5 |
| AC4 | M6 |
| AC5 | M5 |

Table 3: Trace Between Anticipated Changes and Modules

# 8 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.

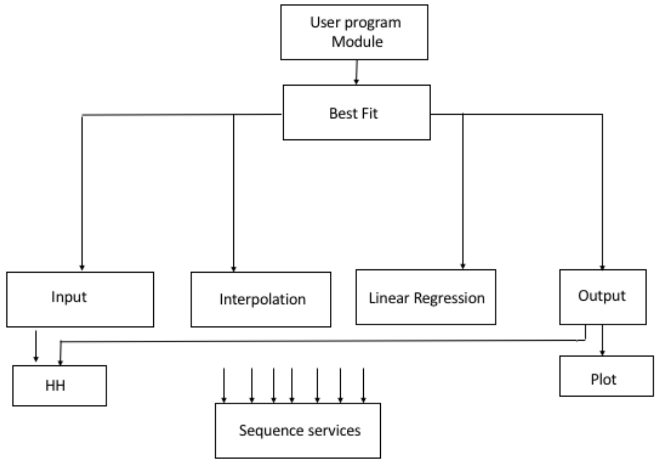


Figure 1: Use hierarchy among modules

# 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.