Module Guide: Stock Prediction System

Renjie Zhang

December 18, 2017

1 Revision History

Date	Version	Notes
01/11/2017 1	1.0	create
02/11/2017 2	1.1	update

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.

AC3: The layout of the plot

AC4: The implementation of the Kernel function

AC5: The implementation of the Price Volatility

AC6: The implementation of the Price Momentum

AC7: The implementation of the Prediction

AC8: The format of the result output

AC9: The change of the distributed system

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.

UC3: There will be another big data platform to replace Spark

UC4: The change of algorithm method

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: Data Input Module

M3: Kernelling Module

M4: Price Volatility Module

M5: Price Momentum Module

M6: Stock Prediction Module

M7: Output Module

M8: Spark Module

M9: Data Plot Module

Level 1	**Level 2**
Hardware-Hiding	
Behaviour-Hiding Module	Main Module Data Input Module Price Volatility Module Price Momentum Module Prediction Module Data Plot Module
Software Decision Module	RDD Module Kernelling Module

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

Secrets: The contents of the required behaviors.

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 Data Input Module (M2)

Secrets: The format and structure of the input data.

Services: Read the dataset file and convert the input data into the data structure used by the input module.

Implemented By: [Stock Prediction System]

6.2.2 Kernelling Module (M3)

Secrets: The Kernelling function implementation.

Services: Convert the dataset from 2D to 3D classification using Kernelling function and parameters.

Implemented By: [Stock Prediction System]

6.2.3 Volatility Module (M4)

Secrets: The Price Volatility and Index Volatility implementation.

Services: Calculate the Price Volatility and Index Volatility by the price and dates from input data

Implemented By: [Stock Prediction System]

6.2.4 Momentum Module (M5)

Secrets: The Price Momentum and Index Momentum implementation.

Services: Calculate the Price Momentum and Index Momentum by the price and dates from input data

Implemented By: [Stock Prediction System]

6.2.5 Prediction Module (M6)

Secrets: The Stock Prediction implementation.

Services: Calculate the result of the prediction based on the Kernelling Function, Price/Index Volatility and Price/Index Momentum

Implemented By: [Stock Prediction System]

6.2.6 Output Module (M7)

Secrets: Display the result of the calculation

Services: Display the result (Increase or Decrease) by short term and long term

Implemented By: [Stock Prediction System]

6.2.7 RDD Module (M8)

Secrets: Transfer data between driver and workers through RDD format into different worker machines and collect data from distributed RDD system back to the driver.

Services: It provides a distributed system to separate the work from one single machine to a set of workers. Driver assigns the data from Input Module to its workers and workers will do the actual computation based on the data and return the results to the driver.

Implemented By: [Stock Prediction System]

6.3 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: –

6.3.1 Plot Module (M9)

Secrets: Display the history of the stock by the plot of the price and date

Services: Generate the plot based on the price and date from the input file

Implemented By: Python library

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	M2
R2	M9
R3	M3,M4,M5,M6,M8
R4	M2
R5	M7

Table 2: Trace Between Requirements and Modules

AC	**Modules**
AC1	M1
AC2	M2
AC3	M9
AC4	M3
AC5	M4
AC6	M5
AC7	M6
AC8	M7
AC9	M8

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.

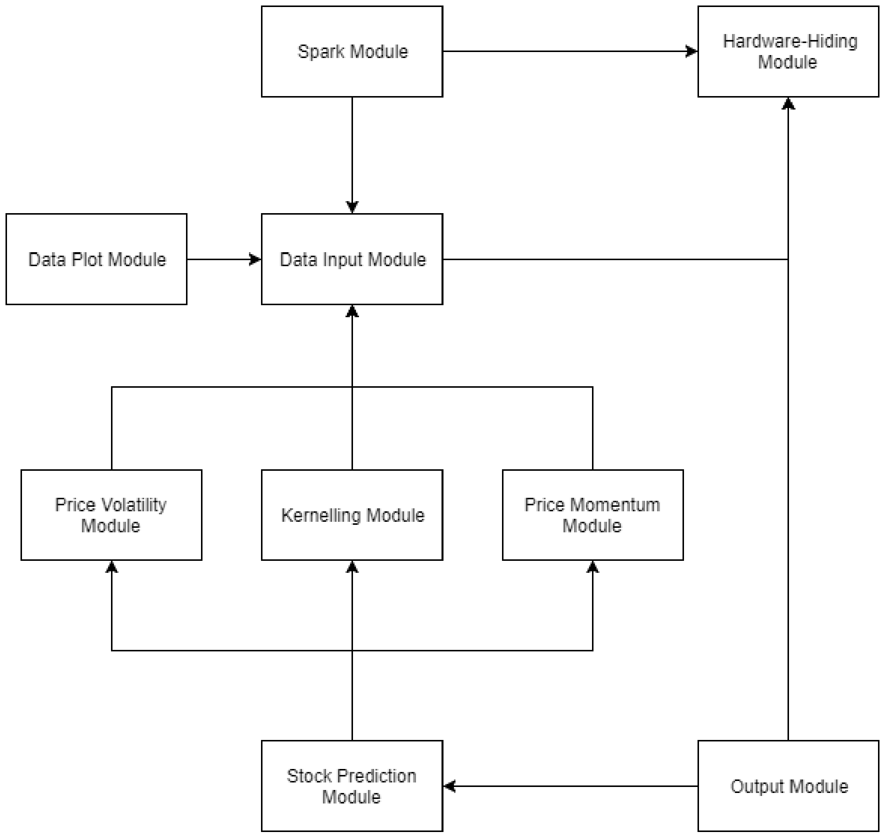

Figure 1: Use hierarchy among modules

[Good start for the design. Remember that you may have to modify the design as you work through the MIS and gain a deeper understanding of how your modules interact. —SS]

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.