# Module Guide for Pot-Pulator

Team 24
Gillian Ford
Juan Moncada
Aaron Billones
Steven Ramundi

January 18, 2023

# 1 Revision History

| Date | Version | Notes |
|---|---|---|
| 2023-01-07 | 1.0 | initial release |

# 2 Reference Material

This section records information for easy reference.

## 2.1 Abbreviations and Acronyms

| symbol | description |
|---|---|
| AC | Anticipated Change |
| DAG | Directed Acyclic Graph |
| M | Module |
| MG | Module Guide |
| OS | Operating System |
| R | Requirement |
| SC | Scientific Computing |
| SRS | Software Requirements Specification |
| ProgName | Explanation of program name |
| UC | Unlikely Change |

# Contents

# List of Tables



# List of Figures

# 3 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 (**?**). 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 layed out by **?**, as follows:

- System details that are likely to change independently should be the secrets of separate modules.
- Each data structure is implemented 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 4 lists the anticipated and unlikely changes of the software requirements. Section 5 summarizes the module decomposition that was constructed according to the likely changes. Section 6 specifies the connections between the software requirements and the modules. Section 7 gives a detailed description of the modules. Section 8 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 9 describes the use relation between modules.

# 4 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 4.1, and unlikely changes are listed in Section 4.2.

## 4.1 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 physical input devices that the user will have available to them.

**AC2:** Calulations for timing conveyor stopage and movement.

**AC3:** Sensors used for verification and validation.(increased accurasy and lower latency)

**AC4:** Communication protocol used to between the individual modules amd main board.

## 4.2 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:** Use of STM32F421i-DISC1 as the main control board and brain of the system.

**UC2:** Use of distance sensing for validation and verification.

**UC3:** Use of Arduino microcontrollers to drive smaller modules.

**UC4:** Tray dispensing delivery method.

**UC5:** Pot dispensing mechanics.

# 5 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:** Front End Module

**M2:** Communication Module

**M3:** Pot Dropping Input Module

**M4:** Pot Dropping Position Module

**M5:** Pot Dropping Stepper Module

**M6:** Pot Dropping Output Module

**M7:** Conveyor Input Module

**M8:** Conveyor Movement Module

**M9:** Tray Dispenser Input Module

**M10:** Tray Dispenser Gantry Module

**M11:** Tray Dispenser Raising Module

**M12:** Tray Dispenser Output Module

**M13:** Verification Analysis Module

**M14:** Verification Output Module

# 6 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.

| Level 1 | Level 2 |
|---|---|
| Hardware-Hiding Module | |
| Behaviour-Hiding Module | Pot Dropping Input Module |
| | Pot Dropping Stepper Module |
| | Pot Dropping Output Module |
| | Conveyor Input Module |
| | Conveyor Movement Module |
| | Tray Dispenser Input Module |
| | Tray Dispenser Gantry Module |
| | Tray Dispenser Raising Module |
| | Tray Dispenser Output Module |
| | Verification Output Module |
| Software Decision Module | Pot dropping Position Module |
| | Verifications Analysis Module |
| | Communication Module |
| | Front End Module |

Table 1: Module Hierarchy

# 7 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. *ProgName* means the module will be implemented by the ProgName 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.

## 7.1 Hardware Hiding Modules (M??)

Due to the hardware focus of this project, there are no hardware hiding module.

**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

## 7.2 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:** –

### 7.2.1 Pot Dropping Input Module (M3)

**Secrets:** Subsystem state and main board command.

**Services:** Take in state and command and allow subsystem action.

**Implemented By:** [Pot-pulator]

**Type of Module:** [Record, Library, Abstract Object, or Abstract Data Type] [Information to include for leaf modules in the decomposition by secrets tree.]

### 7.2.2 Pot Dropping Stepper Module (M5)

**Secrets:** Position of dispensing thread.

**Services:** Drives stepper motors and dispenses pots.

**Implemented By:** [Pot-pulator]

**Type of Module:** [Record, Library, Abstract Object, or Abstract Data Type] [Information to include for leaf modules in the decomposition by secrets tree.]

### 7.2.3 Pot Dropping Output Module (M6)

**Secrets:** State of pot dropping subsystem

**Services:** Serves to maintian the status, errror and readyness of the pot dispensing subsystem as well as communicate this status with main control board.

**Implemented By:** [Pot-pulator]

**Type of Module:** [Record, Library, Abstract Object, or Abstract Data Type] [Information to include for leaf modules in the decomposition by secrets tree.]

#### 7.2.4 Conveyor Input Module (M7)

**Secrets:** Subsystem state and main board command

**Services:** Takes in subsystem state and commnad and allows for system functionaility.

**Implemented By:** [Pot-pulator]

**Type of Module:** [Record, Library, Abstract Object, or Abstract Data Type] [Information to include for leaf modules in the decomposition by secrets tree.]

#### 7.2.5 Conveyor Movement Module (M8)

**Secrets:** Subsystem state and main board command.

**Services:** Serves to drive conveyor motor and creat movement.

**Implemented By:** [Pot-pulator]

**Type of Module:** [Record, Library, Abstract Object, or Abstract Data Type] [Information to include for leaf modules in the decomposition by secrets tree.]

#### 7.2.6 Tray Dispensor Input Module (M9)

**Secrets:** Subsystem state and main board command.

**Services:** Takes in subsystem state and commnad and allows for system functionaility.

**Implemented By:** [Pot-pulator]

**Type of Module:** [Record, Library, Abstract Object, or Abstract Data Type] [Information to include for leaf modules in the decomposition by secrets tree.]

#### 7.2.7 Tray Dispensor Gantry Module (M10)

**Secrets:** Subsystem state and main board command, Calw postion, tray idle position.

**Services:** Serves to drive gantry to dispense tray into Conveyor.

**Implemented By:** [Pot-pulator]

**Type of Module:** [Record, Library, Abstract Object, or Abstract Data Type] [Information to include for leaf modules in the decomposition by secrets tree.]

#### 7.2.8 Tray Dispensor Raising Module (M11)

**Secrets:** Subsystem state and main board command, tray raiser position.

**Services:** Serves to drive stepper motors to raise a tray into its idle position.

**Implemented By:** [Pot-pulator]

**Type of Module:** [Record, Library, Abstract Object, or Abstract Data Type] [Information to include for leaf modules in the decomposition by secrets tree.]

#### 7.2.9 Tray Dispensor Output Module (M12)

**Secrets:** Subsystem state and main board command.

**Services:** Serves to maintian the status, errror and readyness of the tray dispensing subsystem as well as communicate this status with main control board.

**Implemented By:** [Pot-pulator]

**Type of Module:** [Record, Library, Abstract Object, or Abstract Data Type] [Information to include for leaf modules in the decomposition by secrets tree.]

#### 7.2.10 Verification Output Module (M14)

**Secrets:** Verification analysis.

**Services:** Serves to communicate output failure with main board.

**Implemented By:** [Pot-pulator]

**Type of Module:** [Record, Library, Abstract Object, or Abstract Data Type] [Information to include for leaf modules in the decomposition by secrets tree.]

#### 7.2.11 Etc.

### 7.3 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:** –

#### 7.3.1 Front End Module (M1)

**Secrets:** User inputs and individual subsystem status.

**Services:** take in user input and converts into functional command for individual modules.

**Implemented By:** [Pot-pulator]

**Type of Module:** [Record, Library, Abstract Object, or Abstract Data Type] [Information to include for leaf modules in the decomposition by secrets tree.]

#### 7.3.2 Pot Dropping Position Module (M4)

**Secrets:** position of tray within dispensing area.

**Services:** confirms tray position within dispensing area, triggers pot dispensing.

**Implemented By:** [Pot-pulator]

**Type of Module:** [Record, Library, Abstract Object, or Abstract Data Type] [Information to include for leaf modules in the decomposition by secrets tree.]

#### 7.3.3 Verification Analysis Module (M13)

**Secrets:** system functional tolerances, serveral distance readings.

**Services:** Serves to check system output, and to alert if there is an output that does not conform to requirements.

**Implemented By:** [Pot-pulator]

**Type of Module:** [Record, Library, Abstract Object, or Abstract Data Type] [Information to include for leaf modules in the decomposition by secrets tree.]

#### 7.3.4 Communication Module (M2)

**Secrets:** all subsystem states, user inputs.

**Services:** Serves be the communications between user and subsystems. Serves to coordinate action between subsystems.

**Implemented By:** [Pot-pulator]

**Type of Module:** [Record, Library, Abstract Object, or Abstract Data Type] [Information to include for leaf modules in the decomposition by secrets tree.]

#### 7.3.5 Etc.

# 8 Traceability Matrix

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

| Req. | Modules |
|---|---|
| PDR1 | M3 |
| PDR2 | M4 |
| PDR3 | M5 |
| PDR7 | M3 |
| CR1 | M7 |
| CR2 | M8 |
| CR5 | M7 |
| TDR1 | M9 |
| TDR2 | M10 |
| TDR6 | M9 |
| VR1 | M13 |
| VR2 | M14 |

Table 2: Trace Between Requirements and Modules

| AC | Modules |
|---|---|
| AC1 | M1 |
| AC2 | M8 |
| AC3 | M13 |
| AC4 | M2 |

Table 3: Trace Between Anticipated Changes and Modules

# 9 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.

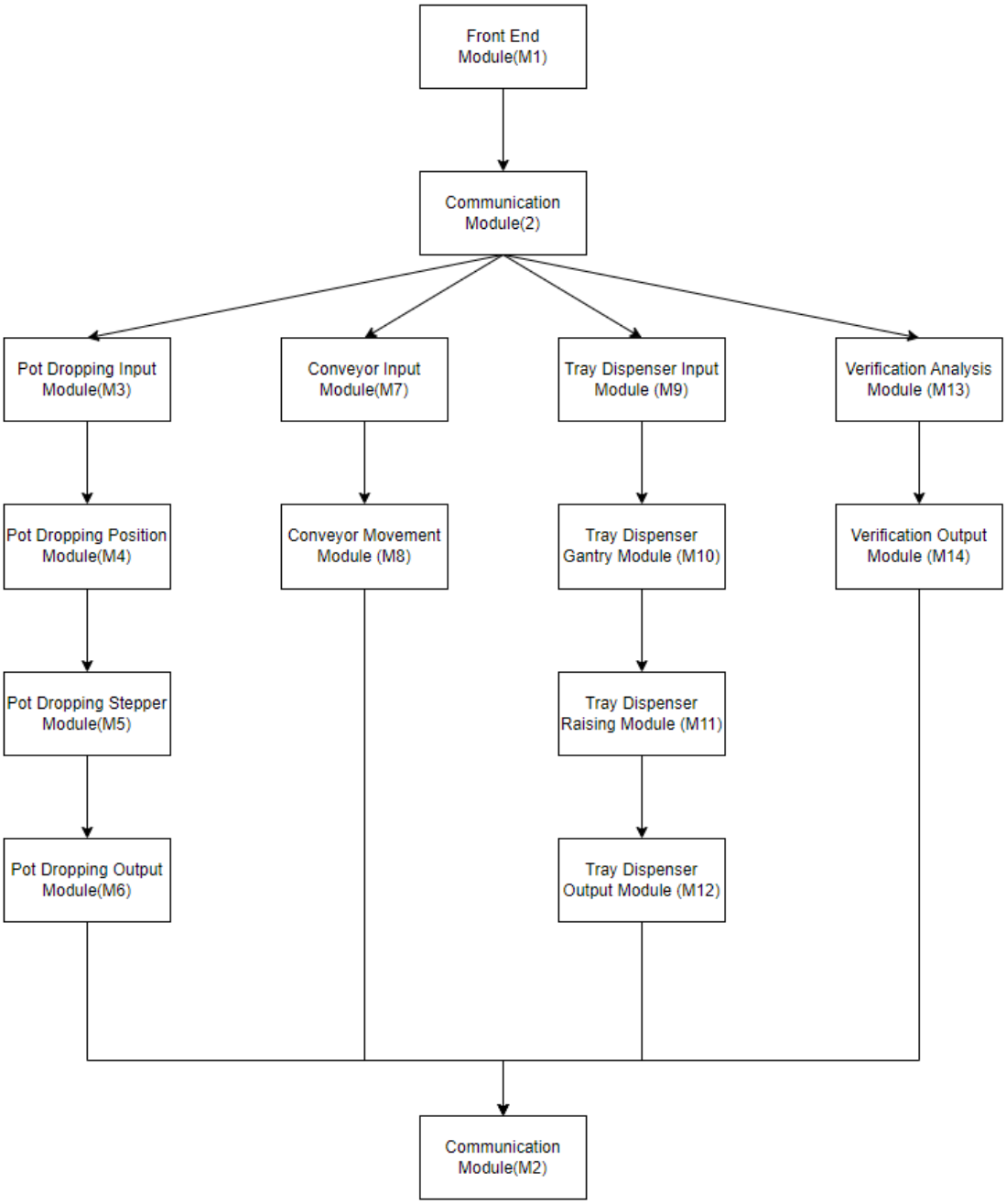


Figure 1: Use hierarchy among modules