Module Guide for SE 4G06, TRON 4TB6

Team 26, STRONE
Jordan Bierbrier
Azriel Gingoyon
Taranjit Lotey
Udeep Shah
Abraham Taha

January 18, 2023

1 Revision History

Date	Version	Notes
Date 1	1.0	Notes
Date 2	1.1	Notes

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
SE 4G06, TRON 4TB6	Explanation of program name
UC	Unlikely Change
[etc. —SS]	[... —SS]

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 **??** lists the anticipated and unlikely changes of the software requirements. Section **??** summarizes the module decomposition that was constructed according to the likely changes. Section **??** specifies the connections between the software requirements and the modules. Section **??** gives a detailed description of the modules. Section **??** 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 **??** 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 **??**, and unlikely changes are listed in Section **??**.

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: Upgraded microcontroller to higher specifications (i.e. more RAM, better processor) will require revisions of the code.

AC2: Sound recognition algorithm will likely improve / be more efficient over time (better algorithm).

AC3: Code will be modified to run on IOS.

AC4: More efficient battery.

AC5: Bluetooth handler may change with newer bluetooth implementations are adopted.

AC6: Code adjusted for higher resolution microphone.

AC7: Upgrading encryption algorithm for user login information.

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: Input data i.e. sound that is processed by software.

UC2: Output data being in the form of haptic feedback.

UC3: Bluetooth being the sole communication method between phone application and microcontroller code.

UC4: Signal processing done on microcontroller as opposed to phone.

5 Module Hierarchy

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

M1: Hardware-Hiding Module

M2: Login Module

M3: Bluetooth Connection Module

M4: Keyword Selection Module

M5: Output Signal Module

M6: Profile Module

M7: Battery Status Module

M8: Sound Classification Module

M9: Bluetooth Communication Module

M10: Microphone Module

Level 1	**Level 2**
Hardware-Hiding Module	
Behaviour-Hiding Module	Login Module
	Bluetooth connection module
	Keyword Selection Module
	Output Signal Module
	Profile Module
	Battery Status Module
Software Decision Module	Sound Classification Module
	Bluetooth Communication Module
	Microphone Module

Table 1: Module Hierarchy

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 **??**.

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. *SE 4G06, TRON 4TB6* means the module will be implemented by the SE 4G06, TRON 4TB6 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??)

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 Login Module (M??)

Secrets: Login authentication algorithm and profile data.

Services: Login capabilities into the application.

Implemented By: Synesthesia Wear

7.2.2 Bluetooth Connection Module (M??)

Secrets: How the Bluetooth connection is established with the application.

Services: Module is used to connect the application to the microcontroller on the wearable device. User clicks the pairing button and the application establishes a connection with an external device.

Implemented By: Arduino Library and Android Studio Library

7.2.3 Keyword Selection Module (M??)

Secrets: How the user's keyword is communicated to the microcontroller/device and their profile data is updated.

Services: Provides user with option of keyword selection and corresponding custom haptic feedback.

Implemented By: Synesthesia Wear

7.2.4 Output Signal Module (M??)

Secrets: How and when the microcontroller communicates with the vibration motor.

Services: Provide haptic feedback to the user through the use of a vibration motor.

Implemented By: Synesthesia Wear and Arduino Library

7.2.5 Profile Module (M??)

Secrets: The database structure which contains the data for the users.

Services: Allows users to login, logout, and update profile settings (keywords, age, name, date of registration).

Implemented By: Synesthesia Wear

7.2.6 Battery Status Module (M??)

Secrets: Algorithm using voltage measurements to estimate the remaining battery life.

Services: Displays remaining battery life of wearable device on the user application. Additionally, alerts user on application when battery below 10% remaining life.

Implemented By: Synesthesia Wear

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 Sound Classification Module (M??)

Secrets: Algorithm used for sound classification. Additionally, filters out sound below specified intensity.

Services: Provides output to vibration motor if keyword is classified.

Implemented By: Synesthesia Wear and Arduino Library

7.3.2 Bluetooth Communication Module (M??)

Secrets: Communication protocol using preexisting connection to communicate between application and microcontroller.

Services: Updates microcontroller with selected keywords, feedback preferences, and battery voltage.

Implemented By: Arduino Library and Android Studio Library

7.3.3 Microphone Module (M??)

Secrets: Method of converting analog signal to digital signal, and relaying this signal to the microcontroller processor.

Services: Provides microcontroller with some of the input data.

Implemented By: Arduino Library

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
R1	M??
R2	M??
R3	M??, M??, M??, M??
R4	M??
R5	M??
R6	M??
R7	M??, M??
R8	M??, M??

Table 2: Trace Between Requirements and Modules

AC	Modules
AC??	M??, M??
AC??	M??
AC??	M??, M??, M??, M??, M??
AC??	M??
AC??	M??, M??
AC??	M??, M??
AC??	M??

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 **??** 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