SE 3XA3: Software Requirements Specification
TouchTime

Team #13, ELM
Evan Ansell - ansellea
Livia Kelle - kellel
Matthew Po - pom

November 10, 2017

Contents

List of Tables

List of Figures

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 (**?**). 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 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.

Table 1: **Revision History**

Date	**Version**	**Notes**
09/11	1.0	Introduction and Anticipated Changes
09/11	1.1	Unlikely Changes
10/11	1.2	Module Decomposition
10/11	1.3	Module Hierarchy, Traceability Matrix
10/11	1.4	Use Hierarchy

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

AC1: The specific hardware on which the software is running (Android Studio Compatible).

AC2: Vibration pattern to distinguish the time

AC3: Watch-face in which the app operates in

AC4: Additional functionality from watch app

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: The app will not be backwards compatible (assumption is that the app will not be run on Android Wear SDK previous to Android 8.0(O))

UC2: There will always be a source of input data external to the software.

UC3: The goal of the system: To tell the time through vibrations

3 Module Hierarchy

This section provides an overview of the module design. Modules are summarized in a hierarchy decomposed by secrets in Table 2. 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 Format Module

M3: Output Vibration Module

M4: Output Display Module

M5: Control Module

Level 1	**Level 2**
Hardware-Hiding Module	
Behaviour-Hiding Module	Input Format Output Vibration Output Display
Software Decision Module	Control

Table 2: Module Hierarchy

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 3.

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. 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.

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 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: Android Studio

5.2.1 Input Format Module (M2)

Secrets: The format and structure of the input data.

Services: Converts and categorizes user input data to match appropriate logic.

Implemented By: Android Studio

5.2.2 Output Vibration Module (M3)

Secrets: Calculated values of input data

Services: Includes methods to send a vibrational response based on user input and current time stored on clock interface

Implemented By: Android Studio

5.2.3 Output Display Module (M4)

Secrets: Information on the hardware display in which the app operates in

Services: Methods which draw and scale the watch face and UI to match the device in which the app is operating in.

Implemented By: Android Studio

5.3 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: Android Studio

5.3.1 Control Module (M5)

Secrets: User input data

Services: Methods which calculate the appropriate response to user input and time at which input was sent and received.

Implemented By: Android Studio

6 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, M4
R2	M2
R3	M3, M5
R4	M5
R5	M3, M5

Table 3: Trace Between Requirements and Modules

AC	Modules
AC1	M1
AC2	M3
AC3	M4
AC4	M1, M3, M4, M5

Table 4: Trace Between Anticipated Changes and Modules

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.

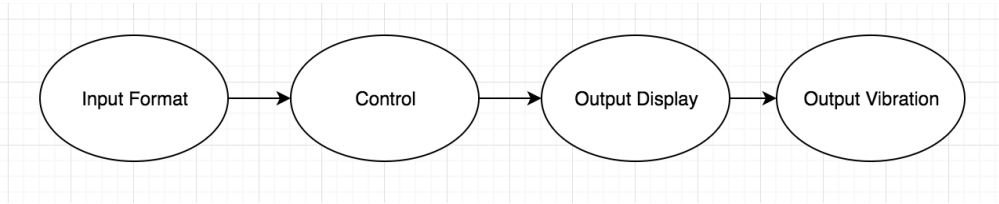

Figure 1: Use hierarchy among modules