

SE 3XA3: Module Guide

ScrumBot

Team 304, ScrumBot
Arkin Modi, modia1
Leon So, sol4
Timothy Choy, choyt2

Last Updated: March 12, 2020

Contents

| | | |
|----------|---|----------|
| 1 | Introduction | 1 |
| 1.1 | Overview | 1 |
| 1.2 | Context | 1 |
| 1.3 | Design Principles | 1 |
| 2 | Anticipated and Unlikely Changes | 2 |
| 2.1 | Anticipated Changes | 2 |
| 2.2 | Unlikely Changes | 2 |
| 3 | Module Hierarchy | 3 |
| 4 | Connection Between Requirements and Design | 3 |
| 5 | Module Decomposition | 3 |
| 5.1 | Hardware Hiding Modules (M1) | 4 |
| 5.2 | Behaviour-Hiding Module | 4 |
| 5.2.1 | ScrumBot Module (M2) | 4 |
| 5.3 | Software Decision Module | 5 |
| 5.3.1 | Meeting List Module (M3) | 5 |
| 5.3.2 | Meeting Types Module (M4) | 5 |
| 5.3.3 | Project List Module (M5) | 5 |
| 5.3.4 | Project Module (M6) | 5 |
| 5.3.5 | Grooming Meeting Module (M7) | 5 |
| 5.3.6 | Generic Dictionary Module (M8) | 5 |
| 5.3.7 | Meeting Module (M9) | 6 |
| 6 | Traceability Matrix | 6 |
| 7 | Use Hierarchy Between Modules | 7 |

List of Tables

| | | |
|---|---|----|
| 1 | Revision History | ii |
| 2 | Module Hierarchy | 4 |
| 3 | Trace Between Requirements and Modules | 6 |
| 4 | Trace Between Anticipated Changes and Modules | 7 |

List of Figures

| | | |
|---|---------------------------------------|---|
| 1 | Use hierarchy among modules | 7 |
|---|---------------------------------------|---|

Table 1: Revision History

| Date | Developer(s) | Change |
|------------------|---------------------|--|
| January 23, 2020 | Arkin Modi | Copy template |
| March 5, 2020 | Leon So | Introduction, Anticipated Changes, Unlikely Changes, Module Hierachy |
| March 10, 2020 | Arkin Modi | Worked on Module Hierarchy |
| March 11, 2020 | Arkin Modi | Worked on the Introduction |
| March 12, 2020 | Arkin Modi | Worked on Introduction, Anticipated and Unlikely Changes, Module Hierarchy, and Module Decomposition |

1 Introduction

1.1 Overview

Scrum is an Agile process framework widely used in industry for managing and coordinating collaborative projects. Scrum follows a highly iterative process and often has heavy customer involvement, therefore it can be often be complex. With Discord being a popular communication tool used by many teams of software developers today, ScrumBot provides a solution that directly integrates the management of a scrum development cycle into the communication channels. ScrumBot will allow for better management and organization of retrospectives, stand-ups, and other scrum/agile stages used by software teams within their routine communication channel. ScrumBot will provide features to add and manage Scrum meetings, as well as to store information relevant to those meetings. ScrumBot will also allow Scrum roles to be assigned to members of the Discord channel.

1.2 Context

Prior to this document, the Software Requirements Specification (SRS) was created to outline all the functional and non-functional requirements this project must satisfy. The purpose of this document is to provide a high-level structure to the implementation of this project by decomposing the idea into modules. The decomposition into modules enables a clearer form to satisfy the requirements of the project.

The Module Interface Specification was created in parallel to this document. It describes the operations that each module shall perform.

1.3 Design Principles

Decomposing a system into modules is a commonly accepted approach to developing software. The Module Guide (MG) developed by (Parnas et al., 1984) specified the modular structure of the system. A module is a work assignment for a programmer or programming team (Parnas et al., 1984). Decomposition of the system into modules is 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.

The 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.

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

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 format of the input data

AC2: The URL used to connect to the hosting server

AC3: The format of the response from the server

AC4: The format of the output data

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:

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

UC3: The system will be interfacing with the Discord application

UC4: The data structure of a meeting

UC5: The data structure of a project

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: ScrumBot Module

M3: Meeting Types Module

M4: Meeting List Module

M5: Project List Module

M6: Project Module

M7: Grooming Meeting Module

M8: Generic Dictionary Module

M9: Meeting Module

M10: Task Module

M11: Task List Module

M12: Sprint Module

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 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.

| Level 1 | Level 2 |
|--------------------------|--|
| Hardware-Hiding Module | |
| Behaviour-Hiding Module | ScrumBot Module |
| Software Decision Module | Meeting List Module Meeting Types Module Project List Module Project Module Grooming Meeting Module Generic Dictionary Module Meeting Module Task Module Task List Module Sprint Module |

Table 2: Module Hierarchy

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

5.2.1 ScrumBot Module (M2)

Secrets: The format and structure of the input data.

Services: Converts the input data into the data structure used by the input parameters module.

Implemented By: scrumbot.py

5.3 Software Decision Module

5.3.1 Meeting List Module (M3)

Secrets:

Services:

Implemented By:

5.3.2 Meeting Types Module (M4)

Secrets:

Services:

Implemented By:

5.3.3 Project List Module (M5)

Secrets:

Services:

Implemented By:

5.3.4 Project Module (M6)

Secrets:

Services:

Implemented By:

5.3.5 Grooming Meeting Module (M7)

Secrets:

Services:

Implemented By:

5.3.6 Generic Dictionary Module (M8)

Secrets: The data structure of a dictionary

Services: Provides basic functionality of a dictionary (add, remove, output data)

Implemented By: dictionary.py

5.3.7 Meeting Module (M9)

Secrets: The data structure of a meeting

Services: Provides a data structure to for a meeting with an initializer, accessor and a mutater

Implemented By: meeting.py

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 | M??, M??, M??, M?? |
| R2 | M??, M?? |
| R3 | M?? |
| R4 | M??, M?? |
| R5 | M??, M??, M??, M??, M??, M?? |
| R6 | M??, M??, M??, M??, M??, M?? |
| R7 | M??, M??, M??, M??, M?? |
| R8 | M??, M??, M??, M??, M?? |
| R9 | M?? |
| R10 | M??, M??, M?? |
| R11 | M??, M??, M??, M?? |

Table 3: Trace Between Requirements and Modules

| AC | Modules |
|------|---------|
| AC?? | M?? |
| AC4 | M?? |
| AC?? | M?? |
| AC?? | M?? |
| AC?? | M?? |
| AC?? | M?? |
| AC?? | M?? |
| AC?? | M?? |
| AC?? | M?? |
| AC?? | M?? |
| AC?? | M?? |
| AC?? | M?? |

Table 4: Trace Between Anticipated Changes and Modules

7 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.