SE 3XA3: Module Guide
Zombie Survival Kit

Team #6, Group 6ix
Mohammad Hussain hussam17
Brian Jonatan jonatans
Shivaansh Prasann prasanns

November 8, 2018

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**
Date 1	1.0	Notes
Date 2	1.1	Notes

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

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.

...

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: Input/Output devices (Input: File, Keyboard, and Mouse, Output: File, Memory, and Screen).

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

...

3 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: Item (Component)

M3: ConsumableItem (Component)

M4: EquipmentItem (Component)

M5: EquipmentManager (Manager)

M6: InventoryManager (Manager)

M7: Interactable (Object)

M8: Enemy (Object)

M9: ItemStore (Object)

M10: InteractableController (Character)

M11: EquipmentUI

M12: EquipmentSlotUI

M13: InventoryUI

M14: InventorySlotUI

M15: CharacterCombat (Character)

M16: CharacterStats (Manager)

M17: PlayerStats (Manager)

M18: ZombieStats (Manager)

M19: Stat (Component)

M20: Zombie (Object)

M21: FirstPersonController (Character)

M22: Gun (Object)

M23: BulletDamage (Objects)

Level 1	Level 2	Level 3
Hardware-Hiding Module	M1	
Behaviour-Hiding Module	M7	M8, M9 (Inherits M7)
	M10	
	M11	
	M12	
	M13	
	M14	
	M15	
	M20	
	M21	
	M22	
	M23	
Software Decision Module	M2	M3, M4 (Inherits M2)
	M5	
	M6	
	M16	M17, M18 (Inherits M16)
	M19	

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 2.

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: –

5.2.1 Interactable (Object) (M7)

Secrets: Serves as a component on an instantiated Game object. Used to differentiate which Game objects can be interacted with by the player.

Services: If a Game object with M7 as a component receives a valid user input, it will produce some output.

Implemented By: Interactable.cs

5.2.2 ItemStore (Object) (M14)

Secrets: Serves as a component on an instantiated Game object; inherits M7. Used to differentiate which Game objects can be stored in the player's inventory.

Services: If a Game object with M14 as a component receives a valid user input, it will store the item in the player's inventory.

Implemented By: ItemStore.cs

5.2.3 InteractableController (Character) (M10)

Secrets: Allows the user to instruct the player to interact with Game objects with M7 or M14 as components through keyboard input.

Services: Converts the keyboard input into an output dictated by M7 or M14.

Implemented By: PickupDropItems.cs

5.2.4 EquipmentUI (M11)

Secrets: Input is determined by any changes that occur in the inventory (i.e. if an item is added or removed from the inventory).

Services: Converts the changes in the inventory to be displayed in the equipment UI.

Implemented By: EquipmentSlot.cs

5.2.5 EquipmentSlotUI (M12)

Secrets: Input is determined by user input through the mouse. Acquires user input if the user clicks on the remove button on any slot in the equipment UI.

Services: Once user input is acquired, the equipment item equipped moves back into the player's inventory and both the Equipment UI and Inventory UI updates accordingly. input parameters module.

Implemented By: EquipmentSlot.cs

5.2.6 InventoryUI (M13)

Secrets: Input is determined by any changes that occur in the inventory (If an item is added or removed from the inventory)

Services: The input converts the changes in the inventory to be displayed in the inventory UI.

Implemented By: InventoryUI.cs

5.2.7 InventorySlotUI (M14)

Secrets: Input is determined by user input through the mouse. Acquires user input if the user clicks on the remove button or the icon button on any slot in the inventory UI.

Services: If an input on the remove button is acquired, the item in the inventory is removed from the inventory and the Inventory UI is updated accordingly. If an input on the icon button is acquired, the item is used in some way (depending on the item) and the inventory UI is updated (E.g. If the item is a consumable item, the item's healthModifier will be added to the player's current health; if the item is an equipment item, the item will be equipped, the player's stats will be updated through the attackModifier and defenceModifier, and the equipment UI will be updated).

Implemented By: InventorySlot.cs

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: –

5.3.1 Item (Component) (M2)

Secrets: Represents the basis of all items that can be used by the player; holds the name of the item, and the sprite icon *not* described in the SRS.

Services: -

Implemented By: Item.cs

5.3.2 ConsumableItem (Component) (M3)

Secrets: Represents the basis of all items classified as a consumable that can be "eaten" by the player; holds the "healthModifier" (integer) that, when the consumable item is "eaten", modifies the player's health by this amount.

Services: -

Implemented By: ConsumableItem.cs

5.3.3 EquipmentItem (Component) (M4)

Secrets: Represents the basis of all items classified as an equipment that can be equipped by the player; holds the "attackModifier" and "defenceModifier" that, when the equipment is equipped, changes the attack and defence stats of the player by these amounts. It also holds "equipSlot" that determines which slot this equipment item belongs to when equipped.

Services: Includes an enum class called equipmentSlot that consists of {Head, Chest, Legs, Primaryhand, Offhand, Feet}. Each instance represents the different areas in which equipment items can be equipped to.

Implemented By: EquipmentItem.cs

5.3.4 EquipmentManager (Manager) (M5)

Secrets: Manages any equipment items that are equipped.

Services: Has an array used to hold any equipped equipment items.

Implemented By: EquipmentManager.cs

5.3.5 InventoryManager (Manager) (M6)

Secrets: Manages any items in the player's inventory.

Services: Contains a list that stores all the items in the inventory.

Implemented By: Inventory.cs

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, M??, M??, M??
R2	
R3	
R4	
R5	
R6	
R7	
R8	
R9	
R10	M??, M??, M??
R11	M??
R12	M??, M??, M??, M??, M??, M??, M??, M??, M??
R13	
R14	
R15	
R16	
R17	
R18	
R19	

Table 2: Trace Between Requirements and Modules

AC	Modules
AC1	M1
AC2	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??

Table 3: 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