# SE 3XA3: Software Requirements Specification
# Staroids

Team 20, Staroids
Moziah San Vicente, 400091284, sanvicem
Eoin Lynagh, 400067675, lynaghe
Jason Nagy, 400055130, nagyj2

November 7, 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 (Parnas et al., 1984). We advocate a decomposition 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.

Our 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.

After completing the first stage of the design, the Software Requirements Specification (SRS), the Module Guide (MG) is developed (Parnas et al., 1984). 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.

Table 1: **Revision History**

| **Date** | **Version** | **Notes** |
|---|---|---|
| Oct 29/18 | 0.1 | Added basic information |
| Nov 7/18 | 0.15 | Added anticipated changes |
| Nov 7/18 | 0.2 | Added some parts of module decomposition |
| Nov 7/18 | 0.25 | Added unlikely changes and module breakdown |
| Nov 7/18 | 0.3 | Added tracability matrix |
| Nov 7/18 | 0.35 | Completed tracability matricies |

- 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 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 Staroids. There are two categories for changes based on the likeliness of the change. Anticipated changes are listed in Section 2.1, and unlikely changes are listed in Section 2.2. Anticipated changes are planned or probable in the forseeable future of Staroids and unlikely changes are changes that are not planned

## 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:** Staroids must be kept up to date with any new operating systems or updates to the supported internet browsers are released.

**AC2:** Enlarging of the playing screen and the scale of all on screen objects.

**AC3:** Support for two players to be active at the same time.

**AC4:** Player firing and destruction sounds.

**AC5:** Collision detection between all on screen sprites.

**AC6:** Relative speed of all projectiles (both player and alien shot)

**AC7:** Speed, locomotion, size and spawning of the alien

**AC8:** The shape and size of asteroids

**AC9:** Player object support for a cooperative or adversarial game mode.

**AC10:** Amount of lives given to the player when the game starts.

### 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:** Controls for the player and menu operations.

**UC2:** The sound module's method of playing sounds and controls over existing sounds.

**UC3:** The destructive property of the large and medium asteroids into 3 asteroids of one size smaller.

**UC4:** The state structure of the core game and how the states transfer into one another.

**UC5:** Method of text dislaying to the screen.

## 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:** appendHead Module

**M3:** Utilities Module

**M4:** Sound Module

**M5:** GameObject Module

**M6:** GameState 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.

| Level 1 | Level 2 |
|---|---|
| Hardware-Hiding Module | M1 |
| Behaviour-Hiding Module | M2<br>M5<br>M6 |
| Software Decision Module | M3<br>M4 |

Table 2: Module Hierarchy

# 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.

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:**

**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

**Secrets:** How the game handles user keyboard input, how the canvas is outlined.

**Services:** Loads scripts onto canvas, therefore giving the user a visual reference for the game to then decide how they would like to interact with it.

**Implemented By:** index.HTML

## 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 Sound Module (M4)

**Secrets:** How sounds are played, how audio files are accessed, and the states of each audio object.

**Services:** Controls a plethora of options for each sound that the game might need including: muting, unmuting, pausing and unpausing, stoping, and checking if the sound is paused.

**Implemented By:** sound.js

### 5.2.2 Utilities Module (M3)

**Secrets:** The values of all constants for the game such as max speed ship size ect, and variables to handle user inputs of pressed keys.

**Services:** Funtions to test for if a key is pressed. The game object consisting of score, lives, asteroids ect, as well as getters and setters for all game object attributes.

**Implemented By:** utilities.js

## 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. Changes in these modules are more likely to be motivated by a desire to improve performance than by externally imposed changes.

**Implemented By:** –

### 5.3.1 GameObject Module (M5)

**Secrets:** Contains base game object, along with all other secondary objects which inheret from it. This includes all the attributes specific to each object.

**Services:** Getters and setters for all game objects, draw functions to put them to the canvas, and interaction functions between objects such as collisions.

**Implemented By:** GameObject.js

### 5.3.2 GameState Module (M6)

**Secrets:** StateMachine for the game

**Services:** Includes data structure and algorithms used in the system that do not provide direct interaction with the user. Changes in these modules are more likely to be motivated by a desire to improve performance than by externally imposed changes.

**Implemented By:** –

# 6 Traceability Matrix

Below shows the traceability matrices for Staroids. These track which requirement is fulfilled by which module and which modules would need to be changed for each anticipated change.

| Req. | Modules |
|---|---|
| R1 | M1, M6 |
| R2 | M6 |
| R3 | M6 |
| R4 | M6 |
| R5 | M6 |
| R6 | M6 |
| R7 | M6, M5, M3 |
| R8 | M5, M3 |
| R9 | M5 |
| R10 | M5 |
| R11 | M5, M6 |
| R12 | M5 |
| R13 | M5 |
| R14 | M5 |
| R15 | M5 |
| R16 | M6 |
| R17 | M5 |
| R18 | M5 |
| R19 | M5, M5, M4 |
| R20 | M5, M5, M4 |
| R21 | M5, M5, M4 |
| R22 | M5, M5, M4 |
| R23 | M5, M4 |

Table 3: Trace Between Requirements and Modules

| AC | Modules |
|---|---|
| AC3 | M3 |
| AC3 | M3 |
| AC4 | M4 |
| AC9 | M5 |
| AC9 | M5 |
| AC9 | M5 |
| AC9 | M5 |
| AC9 | M5 |
| AC10 | M6 |

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.

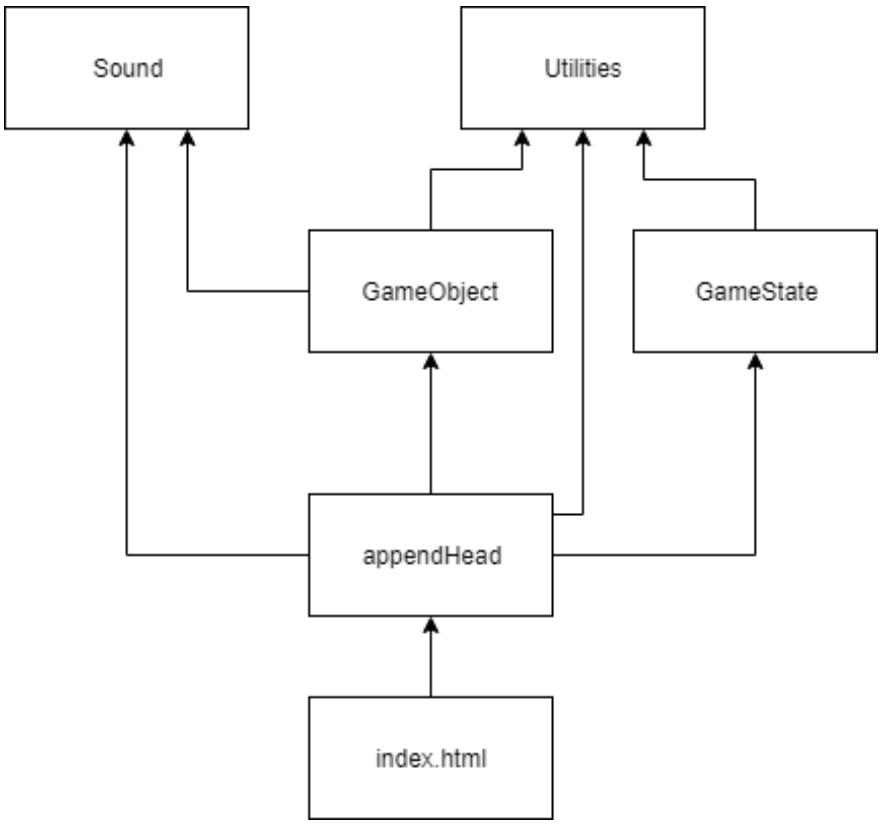


Figure 1: Use hierarchy among modules

# 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.