# QuickShip - System and Software Architecture Description (SSAD)

**DEV-T**

**Victor Fateh (developer)**

**Trinh Nguyen (developer)**

**Emmanuel Mendoza (developer)**

**David Navarro (developer)**

**03/12/2017**

# Version History

| Date | Author | Version | Changes made |
| --- | --- | --- | --- |
| 03/13/17 | Victor Fateh | 1.0 | * Initial Version |

# Table of Contents

System and Software Architecture Description (SSAD) i

Version History ii

Table of Contents iii

Table of Tables iv

Table of Figures v

1. Introduction 1

1.1 Purpose of the SSAD 1

1.2 Status of the SSAD 1

2. System Analysis 2

2.1 System Analysis Overview 2

2.2 System Analysis Rationale 5

3. Technology-Independent Model 6

3.1 Design Overview 6

3.2 Design Rationale 8

4. Technology-Specific System Design 9

4.1 Design Overview 9

4.2 Design Rationale 10

5. Architectural Styles, Patterns and Frameworks 11

# Table of Tables

[Table 1: Actors Summary 2](#_Toc463312298)2

[Table 2: Artifacts and Information Summary 3](#_Toc463312299)3

[Table 3: Process Description 4](#_Toc463312300)4

[Table 4: Typical Course of Action 4](#_Toc463312301)4

[Table 5: Alternate Course of Action 4](#_Toc463312302)4

[Table 6: Exceptional Course of Action 4](#_Toc463312303)4

[Table 7: Hardware Component Description 7](#_Toc463312304)7

[Table 8: Software Component Description 7](#_Toc463312305)7

[Table 9: Supporting Software Component Description 7](#_Toc463312306)7

[Table 10: Design Class Description 8](#_Toc463312307)8

[Table 11: Hardware Component Description 9](#_Toc463312308)9

[Table 12: Software Component Description 9](#_Toc463312309)9

[Table 13: Supporting Software Component Description 10](#_Toc463312310)10

[Table 14: Design Class Description 10](#_Toc463312311)10

[Table 15: Architectural Styles, Patterns, and Frameworks 11](#_Toc463312312)11

# Table of Figures

[Figure 1: System Context Diagram 2](#_Toc463312318)2

[Figure 2: Artifacts and Information Diagram 3](#_Toc463312319)3

[Figure 3: Process Diagram 4](#_Toc463312320)4

[Figure 4: Conceptual Domain Model 6](#_Toc463312321)6

[Figure 5: Hardware Component Class Diagram 6](#_Toc463312322)6

[Figure 6: Software Component Class Diagram 6](#_Toc463312323)6

[Figure 7: Deployment Diagram 6](#_Toc463312324)6

[Figure 8: Supporting Software Component Class Diagram 7](#_Toc463312325)7

[Figure 9: Design Class Diagram 8](#_Toc463312326)8

[Figure 10: Robustness Diagram 8](#_Toc463312327)8

[Figure 11: Sequence Diagram 8](#_Toc463312328)8

[Figure 12: Hardware Component Class Diagram 9](#_Toc463312329)9

[Figure 13: Software Component Class Diagram 9](#_Toc463312330)9

[Figure 14: Deployment Diagram 9](#_Toc463312331)9

[Figure 15: Supporting Software Component Class Diagram 9](#_Toc463312332)9

[Figure 16: Design Class Diagram 10](#_Toc463312333)10

[Figure 17: Process Realization Diagram 10](#_Toc463312334)10

### Introduction

#### Purpose of the SSAD

The purpose of this SSAD is to document the results of the analysis and design of the **QuickShip**. The developers use the SSAD as a reference to the software architecture. **QuickShip** will remain faithful to the architecture specified in this SSAD. Furthermore, the maintainer and client will use this SSAD in order to understand the structure of the software upon final delivery.

#### Status of the SSAD

This SSAD will be continuously updated to sync with the current software development stage. Version 1.0 is the first iteration of this document and therefore will only have the early stages of architecture and design.

### System Analysis

#### System Analysis Overview

The primary purpose of QuickShip is to wirelessly connect two potential players on a singular system to play a game. QuickShip matches two players into one game of battleships, each person will have an individual board hidden from the other participant. The turn based game will give power to each player respectively to place attack moves and affect the opponents ships. The game will update for all individuals after a player submits their choice for an attack. After the UI is updated on both sides, the power of move making switches to the next player and the cycle continues. Moves will be restricted by time limits to reduce the necesity for continous push notifications.

##### System Context

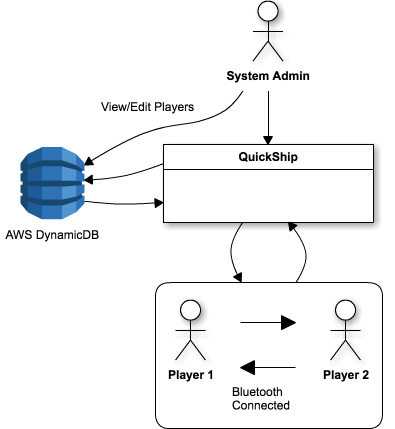


Figure 1: System Context Diagram

Table 1: Actors Summary

| **Actor** | **Description** | **Responsibilities** |
| --- | --- | --- |
| System Admin | Access to live DB | Check validity of users and general debug |
| Player 1 | Actor for one of matched players | Authenticate and play |
| Player 2 | Second actor matched to play | Play game |

##### Artifacts & Information

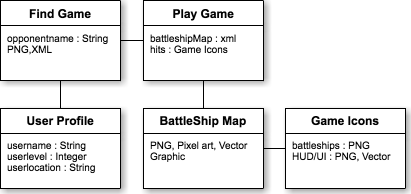


Figure 2: Artifacts and Information Diagram

Table 2: Artifacts and Information Summary

|  |  |
| --- | --- |
| **Artifact** | **Purpose** |
| User Profile | Displays appropriate user's current information to be displayed to other players |
| Finding Game | Main splash screen used as loading area before initializing local connection |
| Battleship Map | Static built grids used and populated by vector icons |
| Game Icons | Android built icons used in game |
| Play Game | Live running screen once connection established. Displays current battleship map with appropriate game icons. |

##### Behavior

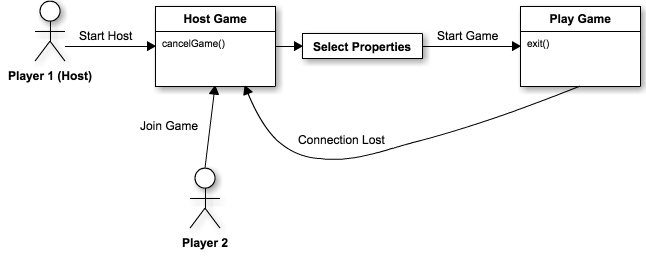


Figure 3: Process Diagram

###### Capability x

Process y

Table 3: Process Description

|  |  |
| --- | --- |
| **Identifier** | Hosting a Game |
| **Purpose** | To allow users to join a game and play together. |
| **Requirements** |  |
| **Development Risks** |  |
| **Pre-conditions** | The host must have Bluetooth or Wifi enabled.  The host must then select to "Host a Game" in the Start Screen |
| **Post-conditions** | The host is then taken to the Game Lobby where players can wait till all players are connected and team configurations have been applied. |

Table 4: Typical Course of Action

|  |  |  |
| --- | --- | --- |
| **Seq#** | **Actor’s Action** | **System’s Response** |
| **1** | User selects to "Host a Game" | Application sets up in Host (server) mode and allows other players to connect. User is now promoted to Host. Host is taken to the Game Lobby state. |
| **2** | Host waits for other players to connect and meet in the Game Lobby | Application stays in standby. |
| **3** | Other players connect to the host game. | Application displays a player has connected. |
| **4** | All players select a team. | Application saves team configurations. |
| **5** | All players select to start game | Application confirms all configurations are set then transitions all players to Setup Gameboard state. |

Table 5: Alternate Course of Action

|  |  |  |
| --- | --- | --- |
| **Seq#** | **Actor’s Action** | **System’s Response** |
| **1** | Host is unable to get enough players. | Application stays in standby. |
| **2** | Host selects "Cancel Game" button. | Application prompts host to verify option |
| **3** | Host selects "Yes" to cancellation prompt. | Application returns to the Start Screen state. |

Table 6: Exceptional Course of Action

|  |  |  |
| --- | --- | --- |
| **Seq#** | **Actor’s Action** | **System’s Response** |
| **1** | Host Bluetooth or Wifi connection is lost. | Application prompts an error prompt and returns to the Start Screen state. |

###### Capability x

Process y

Table 3: Process Description

|  |  |
| --- | --- |
| **Identifier** | Joining a Game |
| **Purpose** | To allow users to connect to a hosted game. |
| **Requirements** |  |
| **Development Risks** |  |
| **Pre-conditions** | The player must have Bluetooth or Wifi enabled.  The player must then select to "Join a Game" in the Start Screen |
| **Post-conditions** | The player is then taken to the Game Lobby where players can wait till all players are connected and team configurations have been applied. |

Table 4: Typical Course of Action

|  |  |  |
| --- | --- | --- |
| **Seq#** | **Actor’s Action** | **System’s Response** |
| **1** | User selects to "Join a Game" | Application queries for Host game. Application joins Host game if available. User is taken to the Game Lobby state and promoted to client. |
| **2** | Client waits for other players to connect and meet in the Game Lobby | Application stays in standby. |
| **3** | Other players connect to the host game. | Application displays a player has connected. |
| **4** | All players select a team. | Host server saves team configurations. |
| **5** | All players select to start game | Host server confirms all configurations are set then transitions all players to Setup Gameboard state. |

Table 5: Alternate Course of Action

|  |  |  |
| --- | --- | --- |
| **Seq#** | **Actor’s Action** | **System’s Response** |
| **1** | Client decides to leave hosted game. | Application stays in standby. |
| **2** | Client selects "Leave Game" button. | Application prompts client to verify option. |
| **3** | Client selects "Yes" to cancellation prompt. | Application returns to the Start Screen state. |

Table 6: Exceptional Course of Action

|  |  |  |
| --- | --- | --- |
| **Seq#** | **Actor’s Action** | **System’s Response** |
| **1** | Client Bluetooth or Wifi connection is lost. | Application prompts an error prompt and returns to the Start Screen state. |

Figure 3: Process Diagram

###### Capability x

Process y

Table 3: Process Description

|  |  |
| --- | --- |
| **Identifier** | Setting Up Game Board |
| **Purpose** | To allow all players to setup their game board before starting the game. |
| **Requirements** |  |
| **Development Risks** |  |
| **Pre-conditions** | Host and Clients have been assigned a team.  All players have selected "Start Game" from the Game Lobby state. |
| **Post-conditions** | All players are taken to the Live Gameplay state. |

Table 4: Typical Course of Action

|  |  |  |
| --- | --- | --- |
| **Seq#** | **Actor’s Action** | **System’s Response** |
| **1** | Player selects a game piece. | Application highlights selected piece. |
| **2** | Player clicks rotate button, if needed. | Game piece is rotated 90 degrees. |
| **3** | Player selects an empty ocean location. | Application loads selected piece into the empty location on the game board. Application then checks if more game pieces are available. If yes; Repeat step 1. If no; move to step 4. |
| **4** | Player selects the "Ready" button | Application prevents any further actions from the player and sets the game board into a "Ready" state. The game board configuration is then transmitted to the host server. |
| **5** | All Players have setup their game board and selected the "Ready" button | Host server initializes the Master Game Board and transitions all Players to the Live Gameplay state. |

Table 5: Alternate Course of Action

|  |  |  |
| --- | --- | --- |
| **Seq#** | **Actor’s Action** | **System’s Response** |
| **1** | Player tries to place a game piece on top of another game piece or out of bounds on the game board. | Application responds with an error prompt. |

Table 6: Exceptional Course of Action

|  |  |  |
| --- | --- | --- |
| **Seq#** | **Actor’s Action** | **System’s Response** |
| **1** | Host or Client Bluetooth or Wifi connection is lost. | Application prompts an error prompt and returns to the Start Screen state or Game Lobby state. |

Figure 3: Process Diagram

###### Capability x

Process y

Table 3: Process Description

|  |  |
| --- | --- |
| **Identifier** | Live Gameplay |
| **Purpose** | To allow all players to now play the game. |
| **Requirements** |  |
| **Development Risks** |  |
| **Pre-conditions** | Host and Clients have setup their game board and have selected the "Ready" button in the Setup Gameboard state. |
| **Post-conditions** | A team wins/loses and all players are taken to the End Game Lobby. |

Table 4: Typical Course of Action

|  |  |  |
| --- | --- | --- |
| **Seq#** | **Actor’s Action** | **System’s Response** |
| **1** | Player selects a square on the opponent's game board via touch screen to engage an attack. | Application records the attack and sends the instructions to the Host Server. |
| **2** | Player waits for all players to make their move. Each move is bound by a timer, all players will have to decide their move within the time restraint. | Application is in standby mode. |
| **3** | Once all players have made their move and or the time restraint has expired. | The Host server will update the Master Game Board and send update instructions to all clients as well as initialize the game play animations for that round. If a team has been declared victorious; move to step 4. If no team has been declared victorious, repeat step 1. |
| **4** | Team has been declared victorious or defeated | Application transitions to the End Game Lobby. |

Table 5: Alternate Course of Action

|  |  |  |
| --- | --- | --- |
| **Seq#** | **Actor’s Action** | **System’s Response** |
| **1** | Player tries to attack a preciously attacked square. | Application responds with a prompt stating the square has already been attacked. |

Table 6: Exceptional Course of Action

|  |  |  |
| --- | --- | --- |
| **Seq#** | **Actor’s Action** | **System’s Response** |
| **1** | Host or Client Bluetooth or Wifi connection is lost. | Application prompts an error prompt and returns to the Start Screen state. |

##### Modes of Operation

<< For projects involving systems that operate in more than one mode this section should contain a description of the system's behavior in each of its modes. More information and example can be found in **ICM EPG> Task: Analyze the Proposed System.** >>

#### System Analysis Rationale

<< This section should list and explain aspects of the analysis that are deemed by the team to be less than obvious or actually counter-intuitive and for which, as a result, there is a high risk that readers will not understand or will misunderstand what is intended. More information and example can be found in **ICM EPG> Task: Analyze the Proposed System.** >>

### Technology-Independent Model

#### Design Overview

##### System Structure

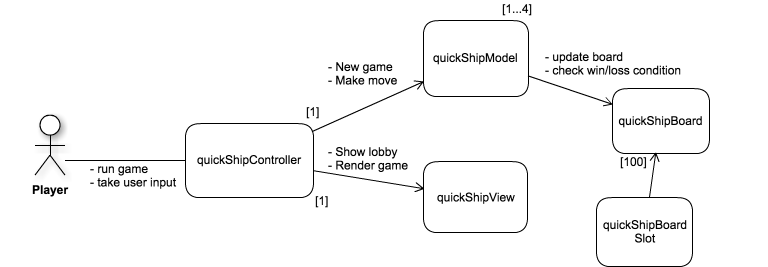


Figure 4: Conceptual Domain Model

Table 7: Hardware Component Description

|  |  |
| --- | --- |
| **Hardware Component** | **Description** |
| Android Device | Android device running API level 16 or a newer |

##### Design Classes

#### Design Rationale

The QuickShip architecture was designed to be as lightweight as possible to maximize playtime. Using PVP sockets reduces the necessity for rapid and constant database queries which consume memory and time. Bluetooth provides a localized means of data exchange between the nearby players. This architecture reduces the stress implemented on the database and allows gameplay and connection issues to be resolved into the Bluetooth breakpoint. Lightweight and uncluttered, this design provides for the most rapid and reliable PVP gameplay experience.

### Technology-Specific System Design

#### Design Overview

##### System Structure

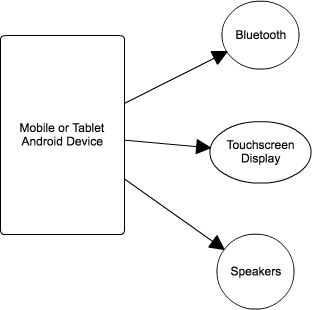


Figure 12: Hardware Component Class Diagram

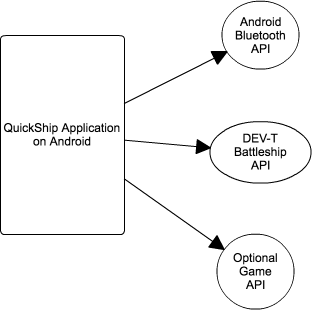


Figure 13: Software Component Class Diagram

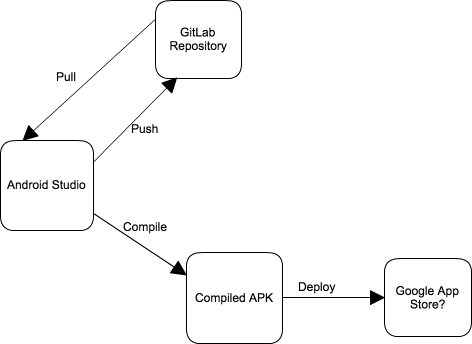


Figure 14: Deployment Diagram

Table 11: Hardware Component Description

|  |  |
| --- | --- |
| **Hardware Component** | **Description** |
| Bluetooth | Uses standard Bluetooth signal to send game data between devices |
| Touchscreen Display | Device Screen must be normal size (470dp x 320dp) to allow viewing complete game content. Touch is also required to operate the game |
| Speakers | Basic mono/stereo speaker to output game audio |

Table 12: Software Component Description

|  |  |
| --- | --- |
| **Software Component** | **Description** |
| Android Bluetooth API | Using free Bluetooth API provided by Google |
| DEV-T Battleship API | Will contain all the backend codes required to run and keep track of game state |
| Optional Game API | May be required to render the game's graphics if Android's standard display API is insufficient |

##### Design Classes

###### <Classes n>

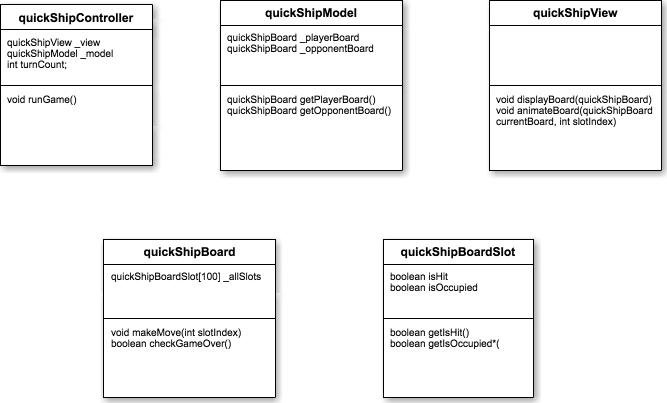


Figure 16: Design Class Diagram

Table 14: Design Class Description

|  |  |  |
| --- | --- | --- |
| **Class** | **Type** | **Description** |
| quickShipController | Controller | Implements thread(s) that keeps track of user input and apply changes to the model |
| quickShipModel | Model | Holds boardgame data of all players to be read by the view or the controller |
| quickShipView | View | Displays game state and data to the screen. Can be done from scratch or use an API to render the visuals |
| quickShipBoard | Model Assets | Holds an array of all individual slots, has functions to modify these slots |
| quickShipBoardSlot | Model Assets | Individual slots that has Boolean to keep track of whether it is hit or occupied |

##### Process Realization

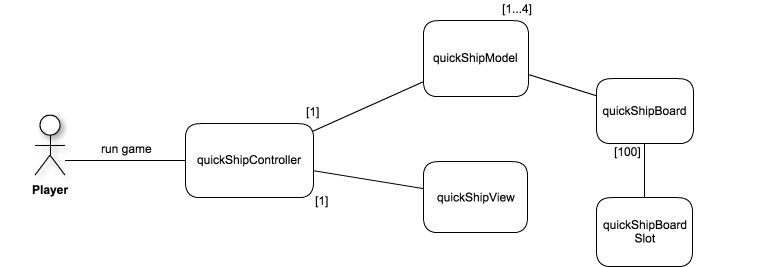


Figure 17: Process Realization Diagram

#### Design Rationale

We hope to implement MVC model to organize the game's classes.

The controller will hold the model of the current game session and maintain the flow of the game. It should implement a thread that checks the status of the game and alert, animate, update, the game as changes are made.

The model will hold data for the board state, what is occupied, what has been hit, etc.

The view will read in the board game in the model and visually represent it on screen.

We hope that this strategy will make the program easier to maintain and implement.

### Architectural Styles, Patterns and Frameworks

Table 15: Architectural Styles, Patterns, and Frameworks

|  |  |  |
| --- | --- | --- |
| **Name** | **Description** | **Benefits, Costs, and Limitations** |
| Android Studio | The main IDE we are using to design the game | The program is free and easy to use. May not be ideal for gaming development |
| Slack | Cloud based communication and project management | Free web-based communication program that tracks Git changes |
| Optional game API | We may choose to use a third party game API if Android Studio Code is limited | These game APIs are usually free but requires some learning |