CENG491
Revised
Design Report

BiBER

Buğra Şirin
Deniz Tav
Engin Fırat
Mert Özer
# Table of Contents

1. Introduction .......................................................................................................................... 6  
   1.1. Problem Definition ........................................................................................................ 6  
   1.2. Purpose .......................................................................................................................... 6  
   1.3. Scope .............................................................................................................................. 6  
   1.4. Overview ....................................................................................................................... 6  
   1.5. Definitions, Acronyms and Abbreviations ................................................................... 7  
   1.6. References ..................................................................................................................... 7  

2. System Overview .................................................................................................................. 7  
   2.1. Core Module .................................................................................................................. 8  
   2.2. BCID Module ................................................................................................................ 8  
   2.3. Game Module ................................................................................................................ 8  
   2.4. Graphical Interpreter Module ...................................................................................... 9  
   2.5. Database Module .......................................................................................................... 9  
   2.6. Logger ............................................................................................................................ 9  

3. Design Considerations ........................................................................................................ 9  
   3.1. Design Assumptions, Dependencies and Constraints .................................................. 9  
      3.1.1. Time Constraints .................................................................................................... 9  
      3.1.2. Resource Constraints ........................................................................................... 9  
      3.1.3. Performance Constraints ....................................................................................... 10  
      3.1.4. Software Constraints ............................................................................................ 10  
      3.1.5. Hardware Constraints ........................................................................................... 10  
   3.2. Design Goals ................................................................................................................ 10  
      3.2.1. Performance .......................................................................................................... 10  
      3.2.2. Reliability ............................................................................................................. 10  
      3.2.3. Functionality ......................................................................................................... 11  
      3.2.4. Usability ............................................................................................................... 11  

4. Data Design ........................................................................................................................ 11  
   4.1. Data Description ........................................................................................................... 11  
   4.2. ER Design ..................................................................................................................... 12  
   4.3. Database Schemas ...................................................................................................... 13  
   4.4. Class Diagrams .......................................................................................................... 14  
      4.4.1. Model Package ...................................................................................................... 14  
      4.4.2. Controller Package ............................................................................................... 15  
   4.5. Data Dictionary ........................................................................................................... 16  

5. System Architecture ............................................................................................................ 17  
   5.1. Architectural design ...................................................................................................... 17  
   5.2. Description of Components ........................................................................................ 18  
   5.2.2. Core Module .......................................................................................................... 18  
      5.2.2.1. Processing Narrative for Core Module ............................................................... 18
6. User Interface Design ................................................................................. 28
   6.1. Overview of User Interface ................................................................. 28
       6.1.1. Login Screen .............................................................................. 28
       6.1.2. Main Screen ............................................................................... 28
           6.1.2.1. Administrator Case ............................................................... 29
           6.1.2.2. Patient Case ....................................................................... 29
       6.1.3. Search Screen ........................................................................... 30
       6.1.4. Registration Screen ................................................................. 31
       6.1.5. Graphic Data Screen ............................................................... 31
       6.1.6. Game Screen ........................................................................... 32
   6.2. Screen Images .................................................................................. 33
7. Detailed Design ........................................................................................................................................ 36

7.1. Core Module ........................................................................................................................................... 36
    7.1.1. Classification ................................................................................................................................. 36
    7.1.2. Definition ......................................................................................................................................... 36
    7.1.3. Responsibilities ............................................................................................................................... 36
    7.1.4. Constraints ....................................................................................................................................... 37
    7.1.5. Compositions .................................................................................................................................... 37
    7.1.6. Uses and Interactions ....................................................................................................................... 37
    7.1.7. Resources ...................................................................................................................................... 38
    7.1.8. Processing ...................................................................................................................................... 38
    7.1.9. Interface / Exports ........................................................................................................................... 38

7.2. BCID Module .......................................................................................................................................... 39
    7.2.1. Classification .................................................................................................................................... 39
    7.2.2. Definition ........................................................................................................................................ 39
    7.2.3. Responsibilities ............................................................................................................................... 39
    7.2.4. Constraints ....................................................................................................................................... 39
    7.2.5. Compositions .................................................................................................................................... 40
    7.2.6. Uses and Interactions ....................................................................................................................... 40
    7.2.7. Resources ...................................................................................................................................... 40
    7.2.8. Processing ...................................................................................................................................... 40
    7.2.9. Interface / Exports ........................................................................................................................... 41

7.3. Game Module ........................................................................................................................................... 41
    7.3.1. Classification .................................................................................................................................... 41
    7.3.2. Definition ........................................................................................................................................ 41
    7.3.3. Responsibilities ............................................................................................................................... 41
    7.3.4. Constraints ....................................................................................................................................... 42
    7.3.5. Compositions .................................................................................................................................... 42
    7.3.6. Uses and Interactions ....................................................................................................................... 42
    7.3.7. Resources ...................................................................................................................................... 42
    7.3.8. Processing ...................................................................................................................................... 42
    7.3.9. Interface / Exports ........................................................................................................................... 43

7.4. Graphical Interpreter Module .................................................................................................................. 43
    7.4.1. Classification .................................................................................................................................... 43
    7.4.2. Definition ....................................................................................................................................... 43
7.4.3. Responsibilities
7.4.4. Constraints
7.4.5. Compositions
7.4.6. Uses and Interactions
7.4.7. Resources
7.4.8. Processing
7.4.9. Interface / Exports
7.5. Database Module
7.5.1. Classification
7.5.2. Definition
7.5.3. Responsibilities
7.5.4. Constraints
7.5.5. Compositions
7.5.6. Uses and Interactions
7.5.7. Resources
7.5.8. Processing
7.5.9. Interface / Exports
7.6. Logger Module
7.6.1. Classification
7.6.2. Definition
7.6.3. Responsibilities
7.6.4. Constraints
7.6.5. Compositions
7.6.6. Uses and Interactions
7.6.7. Resources
7.6.8. Processing
7.6.9. Interface / Exports
8. Time Planning (Gannt Chart)
8.1. Term 1 Gantt Chart
8.2. Term 2 Gantt Chart
9. Libraries and Tools
10. Conclusion
1. Introduction

1.1. Problem Definition

The problem that we attend to fix is rehabilitation of attention deficiency, performance disorders by using neurofeedbacks provided from Brain Computer Interface Devices (BCID). The product is going to help to students, sportsmen and businessman.

1.2. Purpose

This report provides the necessary definitions to conceptualize and further formalize the design of the software, of which its requirements and functionalities were summarized in the previous requirements analysis report. The aim is to provide a guide to a design that could be easily implemented by any designer reading this report.

1.3. Scope

Mindolog will have a hardware part and a software part. Hardware part is the BCID (Brain Computer Interface Device) and software is composed of a simple game which will be controlled by alpha and beta waves of the user, an alpha-beta interpreter and a statistical output graphic.

In this project, we are going to use a brain computer interface that receives alpha and beta waves from the prefrontal cortex of the brain. [1]

The software will save this information to the database and user will be able to see this information. User's brain activities will be displayed while he is playing the game. According to the improvement of the user after the game sessions, the statistical output will be displayed.

1.4. Overview

This document includes detailed design report for Mindolog. First, an overview of the problem and the product are described. Then system overview and design considerations are presented to the audience. After declaring the data design of this project, system architecture will be clarified explicitly. Then user interface design and the detailed design of the Mindolog are explained clearly. And finally, time planning of the project will be given by this document.
1.5. Definitions, Acronyms and Abbreviations

AX-CPT: type of test that wants respondent to memorize the sequence of characters.

BCID: Brain Computer Interface Devices

DDR: Detailed Design Report

EEG: Electroencephalographic

IDR: Initial Design Report

IEEE STD 830-1998: IEEE Recommended Practice for Software Requirements Specifications

GUI: Graphical User Interface

1.6. References


2. System Overview

This system runs according to the data which is provided by the EEG device. In the system the data is processed and modules in the system run appropriately. The system consists of modules hence the system has a modular property. There are six modules namely Core, BCID, Game, Graphical Interpreter, Database and Logger. These modules do not know the source
of the data coming from the core and do not know where the data go which they produce. The general system overview is depicted below:

![System Overview Diagram](image)

**Figure #1 System Overview Diagram**

The explanations of the modules are as follows:

**2.1. Core Module**

This module manages the system. Time management is handled in this module. Moreover all the data flows are accomplished in this module. In this manner other modules cannot communicate with each other. In other words, all the modules act like a black box which exchange data across predefined connections passing through the core module.

**2.2. BCID Module**

This module provides the data coming from the EEG device which is provided by BCID module consisting of two parts. First part is the third-party software provided by the company which produces the EEG device. The other part is a library provided by our sponsor company MINDER. These two software pieces process the meaningless raw data coming from the device and produces the meaningful raw data for our own use.

**2.3. Game Module**

Game uses the incoming data coming from the data flow managed by the core module. Game proceeds by these data and keyboard input.
2.4. Graphical Interpreter Module

This module uses the incoming data coming from the data flow managed by the core module and uses these data to produce output graph which shows the process of the patient.

2.5. Database Module

Saves data coming from the core module to the database and provides data to core module when asked.

2.6. Logger

According to the coming data from Core module, it saves the error message to the log file. The main purpose of this module is to provide technical data when an unexpected situation occurs.

3. Design Considerations

The system is designed with an Object-Oriented paradigm hence each module is presented by a class. Since each class represents a module, all the module classes are implemented by a Singleton Design Pattern. Each module has to communicate only with the Core Module.

3.1. Design Assumptions, Dependencies and Constraints

3.1.1. Time Constraints

The system has to run synchronously with the EEG device. This means that all the modules have to finish their job before EEG device produces new data. For example, if the EEG device produces data in a frequency of 50Hz, the slowest module of the program must accomplish its job in that frequency.

3.1.2. Resource Constraints

Most of the services provided by the modules will be implemented by the help of third party software; therefore, if appropriate software could not be found we have to implement the appropriate one.
3.1.3. Performance Constraints

The system performance is strongly dependent on the EEG device performance because most of the data processed by the modules is provided by the EEG device. Moreover, the interaction between the modules and the core affects the overall performance of the system.

3.1.4. Software Constraints

The system will be implemented by using Java programming language. BCID module will be provided by our sponsor MINDER.

The system consists of a game module. This module needs to render some computer generated graphics. We will use jMonkey graphics engine for this module.

Graphical Interpreter needs to render chart data graphics. To implement this module we will use jFreeChart library.

Database Module interacts with a PostgreSQL DBMS. JDBC will be used to provide appropriate connection between the system and the database.

Logger Module will be implemented by the primitive interfaces of the Java.

3.1.5. Hardware Constraints

There are 128 points in the brain that are suitable for receiving brain waves (alpha, beta, theta, and delta). However, the BCI that we use receives brain waves only from 16 of 128 points. This is the best available technology today. We will be using alpha and beta waves and the correctness of the measurements depends on this hardware.

3.2. Design Goals

3.2.1. Performance

The only part of the system that can be challenging in the sense of performance is the graphical interpreter because it must read and write data concurrently. Design and implementation will be accurate enough to reach the desired performance.

3.2.2. Reliability

Since Mindolog will be a healthcare system and will be keeping data for several patients, we have to design it with minimum faults. Another issue about the reliability is that the data
used and produced must be error-free. We will use various testing strategies at the milestones that we reach while designing the project in order to improve the performance and decrease the number of errors.

### 3.2.3. Functionality

Functionality is another important issue that should be considered. The system serves as not only a statistical database but also a game which helps users to keep their focus on. However, these games must be appropriate enough to measure the brain waves of the patients.

### 3.2.4. Usability

One of the most important design considerations is usability for our project because this product will be used by doctors and patients who can be among any part of the society. Therefore, we should design a simple and understandable interface.

### 4. Data Design

#### 4.1. Data Description

There will be 4 types of data objects in the system namely; user objects, alpha-beta objects, game objects and graph objects.

- **User Objects**: This object will hold user id, password, name, surname, birth date, gender, e-mail and phone information of a patient. Other objects will inherit user information from this data object.

- **Alpha beta objects**: This object will hold amplitude values of each brain wave (alpha and beta) of a patient and the exact time of the measurements. These objects will help to determine the session interval with time. For each object there will be also user id attribute.

- **Game Objects**: This object will hold level number to determine the state of the game when user starts to play. That gives a chance to user to continue from where he/she left the game. User id will also be kept.
- **Graph Objects**: This object will hold session times and user information to show session separately alpha beta values for specified user. While alpha beta objects keep all the history of a patient, these objects are created only when a specific session data is desired to draw the graphs separately.

All of the patients, their alpha and beta values and game information are stored in database. Graph information is not held in the database because it is used only when drawing the graph according to information given. Admin can add patients to database and also can delete. While admin can reach all information of all the patients, every patient can only see their own information.

Alpha-beta objects need user objects by inheriting to store user information. Alpha beta objects are created by game objects. Graph object need user information and alpha beta objects to draw the graph so it needs user objects and alpha beta objects.

### 4.2. ER Design

![Entity Relationship Diagram](image-url)

*Figure #2 Entity Relationship Diagram*
4.3. Database Schemas

- **Administrator Table:**

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Null</th>
<th>Foreign Key</th>
<th>References</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Varchar(30)</td>
<td>No</td>
<td>No</td>
<td>-</td>
</tr>
<tr>
<td>password</td>
<td>Varchar(8)</td>
<td>No</td>
<td>No</td>
<td>-</td>
</tr>
</tbody>
</table>

Administrator Table holds information about the administrator user of Mindolog. It holds the basic attributes of the administrator entity which are name and password.

- **Patient Table:**

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Null</th>
<th>Foreign Key</th>
<th>References</th>
</tr>
</thead>
<tbody>
<tr>
<td>patientID (P.K.)</td>
<td>Integer</td>
<td>No</td>
<td>No</td>
<td>-</td>
</tr>
<tr>
<td>name</td>
<td>Varchar(30)</td>
<td>No</td>
<td>No</td>
<td>-</td>
</tr>
<tr>
<td>password</td>
<td>Varchar(8)</td>
<td>No</td>
<td>No</td>
<td>-</td>
</tr>
<tr>
<td>birthDate</td>
<td>Date</td>
<td>No</td>
<td>No</td>
<td>-</td>
</tr>
<tr>
<td>gender</td>
<td>Varchar(6)</td>
<td>No</td>
<td>No</td>
<td>-</td>
</tr>
<tr>
<td>email</td>
<td>Varchar(30)</td>
<td>No</td>
<td>No</td>
<td>-</td>
</tr>
<tr>
<td>phone</td>
<td>Varchar(30)</td>
<td>No</td>
<td>No</td>
<td>-</td>
</tr>
</tbody>
</table>

Patient Table holds information about the registered user of Mindolog. It holds the basic attributes of the patient entity such as name, password, email etc. Primary key of the patient table is patientID.
• **Alpha-Beta Table:**

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Null</th>
<th>Foreign Key</th>
<th>References</th>
</tr>
</thead>
<tbody>
<tr>
<td>patientID</td>
<td>Integer</td>
<td>No</td>
<td>Yes</td>
<td>Patient</td>
</tr>
<tr>
<td>alphaAmplitude</td>
<td>Float</td>
<td>No</td>
<td>No</td>
<td>-</td>
</tr>
<tr>
<td>betaAmplitude</td>
<td>Float</td>
<td>No</td>
<td>No</td>
<td>-</td>
</tr>
<tr>
<td>measurementTime</td>
<td>Varchar(30)</td>
<td>No</td>
<td>No</td>
<td>-</td>
</tr>
</tbody>
</table>

Alpha-Beta Table holds information about the brain wave values of the registered users' of Mindolog. It holds the basic attributes which are alpha amplitude, beta amplitude and measurement time. patientID is a foreign key which comes from Patient entity.

• **Game Table:**

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Null</th>
<th>Foreign Key</th>
<th>References</th>
</tr>
</thead>
<tbody>
<tr>
<td>patientID</td>
<td>Integer</td>
<td>No</td>
<td>Yes</td>
<td>Patient</td>
</tr>
<tr>
<td>level</td>
<td>Integer</td>
<td>No</td>
<td>No</td>
<td>-</td>
</tr>
</tbody>
</table>

Game Table holds information about the users' game level. patientID is a foreign key which comes from Patient entity.

4.4. **Class Diagrams**

4.4.1. **Model Package**
**Figure #3 Class Diagrams**

### 4.4.2 Controller Package

![Class Diagrams of Controller Package]

**Figure #4 Class Diagrams of Controller Package**
In this package the first two classes will be used to manage the users of the system. Here “Patient Manager” contains the actions that are related to users which will use the system as patients. This class is intended for the use of the admin, i.e. admin will use the methods of this class, and so patient management is done by the administrator. “Authentication Manager” will be used to organize the authentications of the users to the system.

The other three classes seen above will be used for interacting with the database. Alpha-Beta Manager handles storing the wave information of the patient with patientID. The last two classes will also act like an API to publish the data on the user interface. Game Manager starts the game, saves game level for the user with patientID. Graph Manager draws the graphs in the Graph vector.

### 4.5. Data Dictionary

<table>
<thead>
<tr>
<th>DATA</th>
<th>TYPE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>alpha amplitude</td>
<td>float</td>
<td>instantly measured alpha value of the user</td>
</tr>
<tr>
<td>beta amplitude</td>
<td>float</td>
<td>instantly measured beta value of the user</td>
</tr>
<tr>
<td>birth date</td>
<td>string</td>
<td>birth date of the user</td>
</tr>
<tr>
<td>email</td>
<td>string</td>
<td>email of the user</td>
</tr>
<tr>
<td>gender</td>
<td>string</td>
<td>gender of the user</td>
</tr>
<tr>
<td>level</td>
<td>integer</td>
<td>level of the game lastly reached by the user</td>
</tr>
<tr>
<td>measurementTime</td>
<td>string</td>
<td>measurement time of the instantly measured alpha or beta value</td>
</tr>
<tr>
<td>name</td>
<td>string</td>
<td>name of the user</td>
</tr>
<tr>
<td>password</td>
<td>string</td>
<td>password to login the system</td>
</tr>
<tr>
<td>patientID</td>
<td>integer</td>
<td>every patient has a unique id and this data is used to identify patients.</td>
</tr>
<tr>
<td>phone</td>
<td>string</td>
<td>phone number of the user</td>
</tr>
</tbody>
</table>
5. System Architecture

5.1. Architectural design

System will be designed with idea of flow based programming. So we have core module at the center of the system. Core module will be in contact with each other module. There will be 6 modules namely core, BCID, graphical interpreter, database, game and logger. BCID module is responsible from getting alpha beta amplitude values from user’s brain waves. It has sub module that is provided us by the company which generates meaningful values from raw data of alpha beta brain waves. Graphical interpreter module is responsible from creating graphs to users see their alpha beta value progress. It uses the alpha beta values from database to create graph. Database module is responsible from connection of database. It will store and get data from database when core module wants. Database module is important because user will want to see his/her progress session by session to decide whether he/she is going well or not. Therefore all alpha beta values should be saved to the database. Game module will create game environment. It needs level value from database, to create environment according to that level value. Game module also have sub module namely info panel. Info panel will show the concurrent alpha beta values graphically. Logger module is responsible from maintenance of the system. All warning/error messages will be saved to the log files. When one wants to see where the problem of the system is, he/she simply looks to the log files to identify the type of problem. Core module is in the middle of the modules that are described. Core module is responsible from the communication of each module to each other. Here is the overview of the modules;
5.2. Description of Components

5.2.2. Core Module

This module is the most important module of the whole system. All the modules of the system are registered to this module. This module provides a communication interface between the other modules of the system.

Moreover this module includes a tick generator to tick the whole system.

5.2.2.1. Processing Narrative for Core Module

This module ticks all the other modules. The tick generator in it watches the time and it ticks all the modules at the appropriate time. Tick generator also gets data from the Emotiv EPOC in appropriate time intervals.
This module also provides an interface between the modules. For example, when X module wants to call the Y module's method; X calls Y's method through the core module. In addition to this, the data is transferred through the Core Module.

When designing this module we have inspired from a method called Flow-Based Programming. [4][5].

5.2.2.2 Interface Description for Core Module

The interface of the module provides following properties:

Time interface: This interface provides time information to the system. Time interface ticks all the system at the appropriate times.

Data Interface: This interface provides an interface to the system to carry data between the modules. All the data through the system will flow over this interface.

5.2.2.3 Processing Detail for Core Module

This module sees all the other modules as black boxes and this module. All the communications is done over this module. In other words the modules cannot communicate with each other they only communicate with the core module. All the information is that needs to be transmitting somewhere is hold in the core module. So when a module wants data to be transmitted somewhere it writes the data to the appropriate port of the core module. Then core module sends the data to the appropriate module.

5.2.2.4 Dynamic Behavior
5.2.2. BCID Module

BCID module is designed to get alpha and beta values in to the system. With this module user’s alpha and beta waves amplitudes are implemented to the system. In other words, with this module system becomes a system that gets inputs from user’s brain. This part of the system constitutes the most powerful property of the system.

5.2.2.1. Processing Narrative for BCID Module

BCID module is responsible from getting input from user’s brain. It should give double values that have meaning in terms of alpha beta waves amplitudes. These values should be passed to the system once in 250ms. This module passes the incoming alpha beta amplitude values to the core module.
5.2.2.2 Interface Description for BCID Module

BCID module has interface with core module and BCID device. BCID module sends alpha beta amplitude values to the core module. Moreover it gets alpha and beta amplitude values from BCID device. This device generates alpha beta theta and delta values from user’s brain waves. EEGBandData type of structure holds all four values in terms of double type. Layer between the device and our module (which is provided by the company “Minder”), creates these data values from device.

5.2.2.3 Processing Detail for BCID Module

BCID module should get alpha and beta values from device. Our module is subscribed to the library that is provided by the company “Minder”. It gets the data for once in 250ms (since library implemented requires at least this time to get data). It gets a list contains 36 elements consists of alpha beta amplitude values. These values are passed to the core module. With this implementation, concurrent data flow is established.

5.2.2.4 Dynamic Behavior

![Sequence Diagram](image)

**Figure #7 BCID Module Sequence Diagram**

5.2.3. Game Module

This module provides the required routines for the game part of the project. It renders a basic 3D world according to the inputs coming from the keyboard and the EEG device.

This module consists of two sub modules namely Digit-span test module and GraphicsAndSounds module.
AX-CPT Test Module handles the AX-CPT test related tasks. It keeps the test details and provides appropriate information to the GraphicsAndSounds module.

GraphicsAndSounds module uses jMonkey Game Engine to handle some important tasks, such as rendering, playing sounds, a GUI for the player, etc.[6][7]

5.2.3.1. Processing Narrative for Game Module

This module achieves its tasks whenever it is ticked by the Core Module. In every tick of this module, it calculates the position of the game object according to the coming inputs - alpha and beta waves, keyboard inputs - to the module. It plays appropriate sound according to the game state. Moreover an AX-CPT test is also running concurrently.

5.2.3.2 Interface Description for Game Module

The interface of the module provides following properties:

Tick Interface: This interface ticks the whole module. The whole module needs differential time information to do all the calculations correctly so that the passed time after the met, for example position of the game object. This method ticks all the sub modules accordingly.

Alpha Wave Interface: This interface sets the alpha wave information for the current tick.

Beta Wave Interface: This interface sets the beta wave information for the current tick.

Level Interface: This method saves the level of the user to the database.

5.2.3.3 Processing Detail for Game Module

Whenever a tick comes the whole module starts to accomplish required tasks. The tick method of the main modules ticks the two sub modules accordingly.

AX-CPT test module generates a character array in every X seconds. X is determined according to the level of the user. Whenever an array is generated the whole array is sent to the GraphicsAndSounds module to be rendered. After all the estimations done by the user, the estimation array is sent to AX-CPT test module and required calculations are done. According to the estimations the module calculates the results of the user.

GraphicsAndAudio module handles the much of the tasks of the Game Module. jMonkey graphics engine is the most important part of this module. In this module there is a list of the
objects which can be both rendered and played. In every tick all the objects in the list rendered and played accordingly and a check operation is performed to determine whether the AX-CPT test array is ended. If it is ended the estimations will be sent to the AX-CPT test module else the number estimated by the user is pushed back to the estimations array.

5.2.3.4 Dynamic Behavior

![Game Module Sequence Diagram](image)

**Figure #8 Game Module Sequence Diagram**
5.2.4. Graphical Interpreter Module

Graphical Interpreter module is designed to give users a chance to see their alpha/beta values session by session. Users can interpret their data with line charts. By looking to the graph, one can see his/her progress in terms of alpha beta values which mean progress in attention deficiency. Moreover with this module administrators (doctors in our project) can see all patients’ alpha/beta values and compare them. Since in the game screen, there is concurrent alpha beta values are shown this module is also drawing concurrent alpha/beta values.

5.2.4.1. Processing Narrative for Graphical Interpreter Module

Graphical Interpreter module is responsible for the functionality of showing data to the patient of the system and drawing concurrent alpha/beta values. It should give alpha/beta values distinctly by time intervals. Therefore one can easily see whether he/she is progressing or regressing. Administrators can compare patients’ progress with more than 1 couple of line charts. Moreover values got from BCID module should be given to the module to draw graph of concurrent alpha and beta values in game screen.

5.2.4.2. Interface Description for Graphical Interpreter Module

Graphical Interpreter module has interface to get alpha, beta values for the patient or patients. Module wants the data for user with user id. Because it needs data to draw line graph of the patient, it should reach the database to get alpha beta values of each patient. Moreover it needs all data got from BCID module to draw line chart of concurrent alpha and beta values dynamically. Since our product will be based on Flow Based Programming, all interactions with database are done through core module.

5.2.4.3. Processing Detail for Graphical Interpreter Module

Graphical Interpreter module should draw line graph for users. Graph can include one user’s data or more than one user’s data. Basically, it creates a line graph object. It wants the alpha beta values and time interval values from database. Then, it assigns them to the relevant variables of line graph object. Moreover it gets the user’s name from database and assigns it to the graph’s title. For the case that is shown more than one user’s data, lines are assigned to the different colors. Mapping from colors to the users are shown in graph frame. So graph
module needs each user’s information. After getting each user’s information, they are assigned to the relevant variables of line graph object.

Second functionality of the graph interpreter module is to draw graph of alpha beta values that got from BCID module to show to the user his/her concurrent alpha and beta values. It follows the same procedures except it does not need time interval because it is just a line graph that grows with incoming data. Since it is concurrent it is not necessary to specify time.

5.2.4.4 Dynamic Behavior

![Graphical Interpreter Module Sequence Diagram](image)

**Figure #9 Graphical Interpreter Module Sequence Diagram**

5.2.5. Database Module

5.2.5.1. Processing Narrative for Database Module

This module is the interface between the database management system and the core module. Other modules uses database over this module by the connection of core module. In other words, database only communicates with the core module, and the whole system uses database from the core.

5.2.5.2. Interface Description for Database Module

Interface of this module is divided into two packages, namely the Modal Package and the Controller Package. Class diagrams of these packages are presented in the *Class Diagrams* section of *Data Design*. Model Package consists of Patient, Administrator, Alpha-Beta, Game
and Graph classes while Controller Package consists of Patient Manager, Authentication, AlphaBetaManager, GameManager and GraphManager.

5.2.5.3. Processing Detail for Database Module

Model package consists of classes that are used for interacting with the database. They represent the entity structure of the database. When data is requested from the database, related class instances will be created for related database tables and will be filled with the requested information to be used for the requested execution.

Classes defined in the controller package will be used for communicating with the database, i.e. retrieving/updating/deleting/inserting data. The functionality beneath the user interface will be realized by means of this group of classes.

5.2.5.4. Dynamic Behavior

Dynamic behavior of this module is explained in the Data Design section.

5.2.6. Logger Module

Logger module is designed for the maintenance of the whole system. By looking at log files, one can see where the problem is. All error/warning messages are kept in files according to dates. It gets various types of error/warning messages. Moreover all messages have property degrees. Logger module is in communication with core module.

5.2.6.1. Processing Narrative for Logger Module

Logger module is responsible from maintenance of the system. It should give information to the technician about system. It is responsible to detect which message has higher priority. To illustrate, BCID connection problem should have more priority than login problem.

5.2.6.2. Interface Description for Logger Module

Logger module has interface to get data from core module. Core module sends error/warning messages to the module. Logger module has interface with file I/O system. It sends strings and priority information to the file output to write them to the file.
5.2.6.3. Processing Detail for Logger Module

Logger module should save error/warning messages to the log files as described earlier. The module should find the priority of the message whenever module is activated with string (error or warning message). Because some errors could be more important than the others, module needs the priority of the message. When priority of message is found, it is saved to the file with message.

5.2.6.4 Dynamic Behavior

![Sequence Diagram of Logger Module](image)

Figure #10 Logger Module Sequence Diagram

5.3 Design Rationale

We choose 6 part composition because we need 6 main functionalities. These are getting alpha beta values from device, drawing graph from saved alpha beta values, log saving functionality, database communication, and game. 1 extra module created for core module. This type of design gives a chance to add extra modules to the easily after building whole system. Since supporter of us “Minder” stated that there will be more than one games if it will be done accurately, we choose flow based programming. With this type of implementation, we can easily add or remove modules. Other alternative is just connecting each module to the others without having any core module. However it will be costly to add new module to the system when it will be finished. Therefore, we chose this type of module composition.
6. User Interface Design

6.1. Overview of User Interface

6.1.1. Login Screen

When the program gets started, login screen will appear. In login screen, there will be two text boxes to be filled, namely user name and password, and one login button in order to enter the system. After clicking the login button, the system has to check the credentials are validated or not. If they are validated, the system can assign the cookie information by this login screen. If the user is administrator, the cookie contains the information and permissions of the administrator. If the user is the patient, the cookie contains his/her permissions and information.

![Login Screen User Diagram](image)

**Figure #11 Login Screen User Diagram**

6.1.2. Main Screen

There will be two types of main screen which is due to two types of users. The first main screen belongs to administrator user and the second main screen belongs to patients.
6.1.2.1. Administrator Case

In administrator’s case there will be three button namely, registration, show patients data and logout. Registration button will send user to the registration screen which is related with add/edit/delete patient functions that are described below. Show patients data button will send user to the search screen. And logout button will send the user to the login screen.

![Diagram of Administrator User Case]

**Figure #12 Main Screen for Administrator User Diagram**

6.1.2.2. Patient Case

In patient’s case there will be three buttons namely, play game, show my data and logout. As can be understood easily, play game will send user to game screen, show my data button will send user to graphic data screen and logout button will send user to login screen. Show my data button is related with showing user data functions that are specified in part 4.

![Diagram of Patient User Case]

**Figure #13 Main Screen for Patients User Diagram**
6.1.3. Search Screen

There will be one text box to be filled, 2 list fields and three buttons. While entering characters to the text box, all patient entries will be displayed related with the entered characters, in the first list field concurrently. This functionality will use searching patients’ function. When the user select the patient’s entry from the first list field and clicks to the add button, it will be moved to the second list field. If user does not select anything, pop-up screen will appear as warning including “no entry is selected to be added”. When user select patient’s entry from the second list field, and clicks the delete button, the entry will be moved to the first list field. If user does not select anything, pop-up screen will appear as warning including “no entry is selected to be deleted”. Show button will send user to the graph data screen with using patients’ data which are in second list field.

Figure #14 Search Screen User Diagram
6.1.4. Registration Screen

It will be available to administrators only. In this screen, there will be a table which is connected to database with read functions. All patients’ entries will be displayed in the table. There will be 4 buttons namely, add, delete, refresh and save. As can easily be understood, new button will add new patient, delete will delete the selected patient, refresh will refresh the table entries, and save will save the modifications that is done in patients information to database. If user selects one entry from table all data will be filled to the text boxes to give a chance to edit entries.

![Registration Screen User Diagram](image)

Figure #15 Registration Screen User Diagram

6.1.5. Graphic Data Screen

In this screen there will be a graph that is drawn according to selected patient’s id. For administrators, there is a chance to draw more than one patient’s data. For patient users, there will be only their data in graph. There will be two buttons namely, exit and logout. Exit button will send user to the main screen and logout button send user to login screen.
6.1.6. Game Screen

In this screen, there will be three internal frames and two buttons. First frame will show the simultaneous alpha and beta values of user’s brainwaves that is taken from emotive computer brain interface device. Concurrently, in the second internal frame, the game will continue. The game is a simply racing game. It is going to be able to be played by only left and right arrow keys. The cars speed is arranged by the users’ concentration on the game. If she/he does not focus on the game enough, cars speed will increase and vice versa. And in the third frame there will be letters changing concurrently which patients has to pay attention to remember their order to be able to increase his/her attention ability. First button (exit) will send user to the main screen. Second button will save the level of game, which is being played, to the database with write functions and return the main screen.
6.2. Screen Images

6.2.1. Login Screen

Figure #17 Game Screen User Diagram

Related class diagrams are mentioned in the 4. Data Design section on the report.
6.2.2. Main Screens

Welcome to MINDOLOG

- Play the Game
- Show my Data
- Logout

Screenshot #2

Welcome to MINDOLOG

- Registration
- Search / Compare
- Logout

Screenshot #3

6.2.3. Search Screen

Full Patient List

Search List

Search

Screenshot #4
6.2.4. Registration Screen

![Screenshot #5](attachment:image.png)

6.2.5. Game Screen

![Screenshot #6](attachment:image.png)

6.3. Screen Objects and Actions

On all of the screens, there will be Exit bottom object which’s action is returning the main page. And also there will be Log-Out bottom object which’s action is to return the login page. In Registration Screen there will be new delete refresh and save bottom objects which
7. Detailed Design

7.1. Core Module

The Core Module, actually whole of the system works in two states. The first state is called active state and the second is called passive state. The core module is in the active state when the game is playing by the user. In this state the system works in real time. The core module is in passive state whenever the game is not playing and user attempts to do something related with the past. For example, reading his/her past alpha or beta wave information from the database.

Core Modules behaves differently in these two states and the following titles are explained according to these two different states.

7.1.1. Classification

Core Module is a module of the whole system.

7.1.2. Definition

In the active state this module generates ticks for the whole system and provides a communication interface between the modules of the whole system.

In the passive state this module just provides a communication interface between modules. Some services may be invoked according to the user’s inputs.

7.1.3. Responsibilities

This module has two responsibilities. The first responsibility of this module is to generate appropriate ticks for the system. Moreover this module has a responsibility to get data from the BCID module at appropriate times. This data set is then broadcasted to the system. These responsibilities are available whenever the module is in the active state.

The another responsibility of this module is to provide a communication interface for the other modules. All the data flow over the core module hence core module has a
responsibility to receive and transmit data between the modules. This responsibility is valid for the two states of the core module.

7.1.4. Constraints

There is one constraint for the active state of this module. This constraint is about the timing of the system. This module has to generate ticks at appropriate times. The data coming from the BCID device and system's internal time must be consistent with each other. Hence the time phenomena in the system are so important because, all the psychological graphics (alpha and beta waves) and the game object's dynamics are influenced by the wrong differential times.

7.1.5. Compositions

This module composed of the two independent modules namely TickGenerator and DataFlow modules.

7.1.6. Uses and Interactions

Core module interacts with all the other modules of the system. The two sub-modules of this module interact with the system according to the following statements.

The tick generator module generates the tick signal and it is sent to all the other modules while the system is running. All the other modules are started to run after they are ticked by the tick generator. Tick generator sends the differential time (delta time) information to the other modules. This delta time is the time between the previous and current ticks. Then all the physical calculations of the game module and database module are done based on the differential time. The ticking system is active whenever the module is in the active state.

Other sub-module of this module called the Data Flow sub module can be thought as a bridge, at which all the data transfers are done. The connections between the modules are defined before the run of the system. So that whenever a module wants to send data to another, the data is sent through the DataFlow module and whenever this module is invoked the data is sent to appropriate modules according to the pre-defined connection rules. These interactions are valid for the two states of the system.
The two sub modules of this module interacts accordingly however there is no interaction between the two sub-modules of this module. These two sub-modules are independent.

7.1.7. Resources

Data-Flow sub module needs predefined connection rules to construct the communication protocols for the whole system. Appropriate communication rules shall be provided to the system before the run.

Moreover the TickGenerator sub module needs to handle the time. This will be done by using the java system library. Java's System class has a method called currentTimeMillis() in it. This method provides us the current time of the system in the milliseconds. By using this information this sub module can generate appropriate differential times for the system.

7.1.8. Processing

There are two main tasks to be satisfied for the core module. These two tasks are done by the sub modules of this module namely DataFlow module and TickGenerator module.

TickGenerator module reads the holds the time information in it. Whenever all the modules have done their job and appropriate time is passed after the last tick signal, the tick generator ticks the whole system again and sends the differential time information to the other modules which is simply the difference between the current time and last tick time. TickGenerator module is active when the Core Module is in active state.

Whenever the DataFlow module is invoked over a connection the DataFlow module sends the data to the appropriate modules by invoking them. So by these processes the data flow between the modules are satisfied. According to this communication protocol the other modules of the whole system do not interact with each other, they can only interact with the core module. This pre-defined communication protocol gain the modularity property to the whole system. DataFlow module is active for the two states of the system.

7.1.9. Interface / Exports

This module provides an interface to the whole of the system. This interface consists of one method.
The DataFlow sub module provides `sendData()` method. By the help of this method other modules can send data to the other modules over the Core Module. The data to be sent consists of a link id and data itself. With link id information the DataFlow module can send the data to the appropriate module.

Moreover this module provides exports for the system. The TickGenerator sub module provides the most important export for the whole system. Whenever a tick is generated, the differential time information is sent to the all modules over appropriate interfaces of the modules. In other words all the other modules are ticked by the help of this export.

### 7.2. BCID Module

#### 7.2.1. Classification

BCID Module is the sub module of the system.

#### 7.2.2. Definition

BCID module is for getting alpha beta brain wave’s amplitude from BCID device. It gives these data to the core module, to send it relevant modules.

#### 7.2.3. Responsibilities

This module is responsible from brain interface between user’s brain and program. It should give the alpha and beta wave’s magnitudes for 250ms interval. This module is also responsible from establishing connection of device to the system.

#### 7.2.4. Constraints

There are 3 basic variables from predefined data types for module. These are from `EEGBandDataListener`, `EEGSampleBandData` and `EEGBandData`. `EEGBandDataListener` is a class from epoclib library that is provided us by our sponsor company. It informs our module with giving object from type of `EEGSampleBandData`. `EEGSampleBandData` consists of list of `EEGBandData`. `EEGBandData` is also class from epoclib library. It consists of 4 variables namely, alphaValue, betaValue, thetaValue and deltaValue. For 250ms period time it gives a list of 5 elements that is from `EEGBandData` which is included by an object of `EEGSampleBandData`. alphaValue, betaValue, thetaValue and deltaValue are type of double in java. There is also one more list consists of only alphaValue and betaValue. Since the
system only uses alpha and beta amplitudes of brain, there is no need for other brain wave amplitudes.

7.2.5. Compositions

There are basically two subcomponents. One of them is responsible from getting data from device with using epoclib library specified functions and data types. Other is responsible from eliminating theta and delta wave’s amplitude values. The one responsible from getting data from device is connected to the second one. It gives the list of values that is from type of EEGBandData.

7.2.6. Uses and Interactions

First module, that is responsible from getting alpha beta wave’s amplitude, has interaction with the BCI device itself. It creates EEGBandDataListener to listen the data coming from device. It gets data as a list of alpha, beta, theta and delta amplitude values.

First module also has interaction with second module that is responsible from eliminating unnecessary values from list. First module gives the list that contains alphaValue, betaValue, thetaValue and deltaValue to the second module.

7.2.7. Resources

Epoclib library is needed for this module. Because it includes listener classes, to get data from device, module needs this library. This library is provided us by our sponsor company “Minder” as stated before. It does a fast Fourier transform on raw data of BCID and gives meaningful double values. Library needs at least 250ms to get and interpret data coming from device. Therefore minimum time interval to get data is 250ms.

7.2.8. Processing

As it is described earlier, module is responsible from establishing connection between device and system. Firstly it calls the function connectToBCI() that is included from epoclib library. It returns a bool. If it is false then module pops up a warning message, and sends the log message to the core module with relevant id. Core module is expected to send it to the logger module to save the message to the log file.
Second responsibility of the module is getting alpha beta wave’s amplitude values. It gets the data simply with the function of EEGBandDataListener object, informEEGBandData. It gets the concurrent data from device in a list. After that, the list in an object that is type of SampleBandData class, is sent to the sub module to eliminate its unnecessary wave amplitude values. After getting list, sub module creates new list that contains only alpha beta wave amplitude values and sends it to the core module with relevant id.

7.2.9. Interface / Exports

The BCID module has interface with core module. It can give a list that contains alpha and beta amplitude values and relevant id with it. Another possibility is to give log message and relevant id with it. List consists of elements that are type of some kind of EEGBandData. But this time, new EEGBandData, namely AlphaBetaAmplitude, consists of only two double that is alphaValue and betaValue. Second element is id. It is simply an integer that represents, what will be done with this data. It gives an idea to core module about what to do with data.

7.3. Game Module

7.3.1. Classification

Game module is a module of the whole system.

7.3.2. Definition

The purpose of this module is to provide a game world to the user and reform this world according to the user's psychological state and psychological tests running.

7.3.3. Responsibilities

This module is responsible to accomplish some psychological tests and rendering and playing sounds. This module renders the game objects and simple 3D world. The game object's position is determined according to the alpha and beta waves and to the keyboard inputs. The appropriate sounds are also played accordingly. The game module calculates the level of the user according to some psychological surveys and evaluations. According to the users’ level, simple or hard games will be prepared by this module.
7.3.4. Constraints

This module can consume higher times while accomplishing its jobs. There is a time constraint on the whole system; all the system must be run synchronized with the BCID module. So this module must be implemented optimally. There are no any other constraints on this module.

7.3.5. Compositions

This module is consisted of two sub-modules. First one is the AX-CPT Test Module which is responsible to run this psychological test. Other module is the GraphicsAndSounds module. This module is responsible for render the 3D game world and playing appropriate sounds.

7.3.6. Uses and Interactions

Game Module only interacts with the Core Module over the whole system. Core Module invokes the tick method of this module and feeds this module by alpha and beta wave information in every tick. At the end of the game GameModule sends the level information of the player to the Core Module.

The sub-modules of this module interact strongly with each other. AX-CPT Test Module sends the character array to the GraphicsAndSounds module to be rendered. The test is run in the GraphicsAdSounds module. This module keeps the estimations done by the user. When the required number of estimations are done the estimations array is sent to the AX-CPT Test Module. This module checks whether the estimations done are correct or not. According to the result a score for the player will be calculated.

7.3.7. Resources

A software library called jMonkey is used to implement most important functionalities of this module. jMonkey is a game engine and it has built-in physics engine, network engine, audio engine and a graphical user interface. Simple physics of the car in the game is implemented using built-in physics engine of the jMonkey. Moreover the game menu is implemented by the help of the built-in GUI property of the engine.

7.3.8. Processing

The game module simply accomplished its duties with helps of its sub-modules. The AX-CPT Test Module generates character arrays randomly. The length of this array is determined by
the level of the user. The GraphicsAndSounds module holds all of the objects in an array like structure that can be rendered or played. In every tick this module is render the new frame by iterating through the object array.

7.3.9. Interface / Exports

This module provides an interface to the core module. This interface consists of two methods.

The first method is the **tick (float dT)** method. Whenever this method is invoked the whole module starts to accomplish all of its duties such as doing physical calculations, rendering objects, playing sounds.

The second method of this module is the **dataset ()** method. By using this method the core module can send data to this module. Data will be hold in a special structure. This structure consists of a link id and the data itself. With the given link id the data can be sent to the appropriate modules.

The exports of this module can be described by the following statements:

This module needs to write some data to the other modules such as Database Module. Whenever this need occurs this module uses the appropriate interface of the Core Module. The appropriate module of the core module is invoked by two arguments first is the link id and the second one is the data itself.

Moreover whenever this module finishes all of its jobs, this module sends a SUCCESS signal to the core module.

7.4. Graphical Interpreter Module

7.4.1. Classification

Graphical Interpreter Module is a sub module of the system.

7.4.2. Definition

Graphical Interpreter Module is designed to give a chance to see alpha beta amplitude values to patients and doctors. In other words, regular users and administrators are able to see the alpha/beta values of history.
7.4.3. Responsibilities

Graphical Interpreter Module is responsible for drawing line graphs according to the values that are kept in database. Moreover, it should draw concurrent alpha/beta wave’s amplitude values that are coming from BCID module. This line graph is shown in the game screen.

7.4.4. Constraints

There are two modes of running this module; drawing recent session’s values and concurrent wave amplitude values. For first type, there are 2 basic data types namely, Graph class and SessionValue structure. Graph object is created from user name, age and two vectors from the type of SessionValue. User name is type of string. Age is type of integer. Other two vectors hold the alpha and beta values for each session separately. Name of these vectors are alpha and beta. SessionValue structure includes two variable namely, session_id and wave value. Session_id is as easily understood represents the session number. Wave value is for holding alpha or beta amplitude values.

LineChart object is created from graph object. LineChart object is defined in jFreeChart library.

7.4.5. Compositions

There are basically 3 sub component, namely converter, concurrent graph and historical graph. Each sub component has different roles. Converter subcomponent is responsible from converting incoming data, which is type of vector of AlphaBetaValue, to two vectors namely, alpha and beta with having SessionValue elements. According to the time values of incoming data, it detects which data belongs to which session. Concurrent graph sub component is responsible from showing concurrent alpha beta values of user. It does not need to specify user name or session id, so it does not need the value that comes from converter sub component.

Historical graph is responsible from showing older session values of user. It needs the values come from Converter subcomponent. It needs user name and age to specify them in the line graph.
7.4.6. Uses and Interactions

Converter has interaction with core module. It takes the vector of AlphaBetaValues that contains alphaValue and betaValue separately. Historical graph sub component has interaction with Converter sub component. It needs the value that is supplied from converter sub component. Converter gives vector of SessionValues to it. It is most crucial thing to draw the line graphs.

Concurrent graph sub components have interaction with core module, to get relevant AlphaBetaValues. It does not need the values that are converted in Converter sub component. It just takes the vector of AlphaBetaValues and draws two different graphs from it.

7.4.7. Resources

JFreeChart library is needed to draw line charts. LineChart class of this library is used for drawing graphs.

7.4.8. Processing

When data comes from core module, according to graph type wanted from core module, data is sent to the Converter or Concurrent graph sub module. If graph_type is 1 which means it should draw concurrent graph, data is sent to the concurrent graph sub module. Otherwise it is sent to the Converter sub module. For graph_type having 1 case, data directly is sent to concurrent graph sub module. Concurrent sub module has its own vectors that are recent alpha beta values came from BCID Module. If its size is greater than 20, it pops the oldest value and pushes the new one.

For case 2, data is sent to the Converter sub component. A vector having time values and amplitude values come into the Converter sub component. According to the time values, module determines which values are in the same session. For each session it creates overall value by taking mean of them. All session values are pushed into the relevant vector. There are two types of vectors namely, alpha and beta. These vectors have type of SessionValue which contains session id as integer and alpha or beta value as integer. After getting two different vectors, LineChart object is created and all relevant values are filled to the LineChart object. User name and age are given as title to the graph.
7.4.9. Interface / Exports

Graphical Interpreter Module has interface with core module only. It gets values of alpha beta wave magnitudes. According to graph type it can have time value inside it or not, it can have only 250ms data or whole data saved to the database.

7.5. Database Module

7.5.1. Classification

Database module is a module of the whole system.

7.5.2. Definition

This module is the interface between the database management system and the core module. Other modules uses database over this module by the connection of core module. In other words, database only communicates with the core module, and the whole system uses database from the core.

7.5.3. Responsibilities

The purpose of this module is to make it possible to communicate the other modules with the database. It manages the receiving and storing information to the database according to the signals coming from the other modules.

7.5.4. Constraints

This module is completely dependent on the other modules. It assumes the core module always transforms the correct information to be kept or to be delivered.

7.5.5. Compositions

Interface of this module is divided into two packages, namely the Modal Package and the Controller Package. Model Package consists of Patient, Administrator, Alpha-Beta, Game and Graph classes while Controller Package consists of Patient Manager, Authentication, AlphaBetaManager, GameManager and GraphManager.

7.5.6. Uses and Interactions

Model package consists of classes that are used for interacting with the database. They represent the entity structure of the database. When data is requested from the database, related class instances will be created for related database tables and will be filled with the
requested information to be used for the requested execution. Classes defined in the controller package will be used for communicating with the database, i.e. retrieving/updating/deleting/inserting data. The functionality beneath the user interface will be realized by means of this group of classes.

7.5.7. Resources

Database Module interacts with a PostgreSQL DBMS. PostgreSQL is a powerful, open source object-relational database system.

JDBC(The Java Database Connectivity API) will be used to provide an appropriate connection between the system and the database. It is the industry standard for database-independent connectivity between the Java programming language and a wide range of databases, SQL databases and other tabular data sources.

7.5.8. Processing

The component performs the duties over the objects which consist of the Model Package. There will be 4 types of data objects in the system namely; user objects, alpha-beta objects, game objects and graph objects as explained before. Patient plays game, game creates alpha-beta objects, graph object reads these alpha-beta objects and users can view the graphs. To enable this, these objects must communicate each other. Here is the definition of the objects’ methods:

Patients has getID() method because all the other objects certainly needs the id of the patient to relate all the objects with a specific patient. Alpha-Beta objects have getAlphaAmplitude() and getBetaAmplitude() methods because Graph objects need these values in order to draw the graph. Game objects has getLevel() method to enable the user continue the game from where he/she left off. Graph objects have setStartTime(string) and setEndTime(string) methods to specify the time interval of the alpha and beta values that is desired for the graph to be drawn.

7.5.9. Interface / Exports

The set of services provided by this module are defined in the Controller Package of the module which includes PatientManager, Authentication, AlphaBetaManager, GameManager and GraphManager. Here is the description of these classes:
PatientManager primarily manages the self-explanatory services which are addPatient(), deletePatient() and updatePatient(). Moreover, it searches among the current patients by name getPatientByName() method so make it possible to list the search results. It also has a getPatientByID() method which does the same thing with ID instead of name.

Authentication manages the login/logout services by comparing the given data with the current data. Besides the login () and logout () methods, it has Authenticate (user) method which gives login right to a specific user, possibly new user.

AlphaBetaManager has saveAlphaAmplitude() and saveBetaAmplitude() methods to save the related values for the patient with a specific id which is also hold in this class as an attribute. It further has a saveMeasurementTime() method, records the time for the use of a graph object for future use.

GameManager has startGame() and saveGame(int) methods which are self-explanatory. saveGame(int) method takes the level value as argument and saves it for a specific user whose id is also hold in this class as an attribute.

GraphManager has drawGraph() attribute which uses the information it has as its attribute graphVector to draw the desired graph.

7.6. Logger Module

7.6.1. Classification

Logger module is sub module of the system.

7.6.2. Definition

Logger module gives a chance to technicians to see the problem of the system by supplying log files to them. By exploring these files, one can see easily which warning message is given a lot. Then it can easily be predicted where the problem is.

7.6.3. Responsibilities

Logger module is responsible from saving all warning and error messages to the log files. It gives all messages priority ids. These priority ids are specified according to importance of the warning or error message.
7.6.4. Constraints

There is one data type for the Logger module. It is from type of String in java. It is for holding error/or warning messages. Moreover there will be a static array of string and integers to hold all types of error/warning messages with priority ids.

7.6.5. Compositions

Since there is only one component, it is the main component of the sub module.

7.6.6. Uses and Interactions

Module gets warning or error message as string. It is the only interaction between core module and Logger Module.

7.6.7. Resources

Since Java file I/O operations are needed, File class of java.io.File library is needed.

7.6.8. Processing

When message comes to the module, it is checked with the array that is specified before, and according to priority id found in array, it is saved to the file that has present date. If there is no log file for present date, first it is created. After writing operation it is closed.

7.6.9. Interface / Exports

Logger module has only interface with core module. It gets error/warning message as data type of string. After saving it to the log file, it sends Boolean true value to the core module.

Related class diagrams are mentioned in the 4.Data Design section on the report.

8. Time Planning (Gannt Chart)

![Figure #18 Estimation Diagram]
### 8.1. Term I Gantt Chart

<table>
<thead>
<tr>
<th>Number</th>
<th>Task</th>
<th>Resource</th>
<th>Start</th>
<th>End</th>
<th>Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Planning</td>
<td>1/30/2010</td>
<td>11/9/2010</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>1.1</td>
<td>Definition of the Problem</td>
<td>1/30/2010</td>
<td>11/2/2010</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1.2</td>
<td>Project Survey</td>
<td>1/1/2010</td>
<td>11/4/2010</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>1.3</td>
<td>Project Proposal</td>
<td>1/30/2010</td>
<td>11/5/2010</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Analysis</td>
<td>1/32/2010</td>
<td>12/5/2010</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>2.1</td>
<td>Analytical Requirements</td>
<td>1/32/2010</td>
<td>11/18/2010</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>2.2</td>
<td>Determining The Problem</td>
<td>1/38/2010</td>
<td>11/19/2010</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>2.4</td>
<td>Functional Modeling</td>
<td>1/20/2010</td>
<td>12/1/2010</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>2.5</td>
<td>Data Dictionary</td>
<td>1/24/2010</td>
<td>12/1/2010</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>2.6</td>
<td>Writing the Report</td>
<td>12/2/2010</td>
<td>12/5/2010</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Design</td>
<td>12/16/2010</td>
<td>1/4/2011</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>3.1</td>
<td>Architectural Design</td>
<td>12/16/2010</td>
<td>12/26/2010</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>3.2</td>
<td>Patient User Interface Design</td>
<td>12/20/2010</td>
<td>1/22/2010</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>3.3</td>
<td>Administrator User-Interface Design</td>
<td>12/20/2010</td>
<td>1/22/2010</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>3.4</td>
<td>Component Level Design</td>
<td>12/23/2010</td>
<td>12/26/2010</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3.5</td>
<td>Prototype Implementation-1</td>
<td>12/23/2010</td>
<td>12/27/2010</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>3.6</td>
<td>Planning About Prototype</td>
<td>12/27/2010</td>
<td>12/27/2010</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>3.10</td>
<td>Component Level Design-2</td>
<td>12/29/2010</td>
<td>1/2/2011</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4.2</td>
<td>Prototype Demo</td>
<td>1/18/2011</td>
<td>1/23/2011</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5.1</td>
<td>Implementation</td>
<td>2/17/2011</td>
<td>4/17/2011</td>
<td>42</td>
<td></td>
</tr>
<tr>
<td>5.3</td>
<td>Unit Testing</td>
<td>4/20/2011</td>
<td>4/29/2011</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>5.7</td>
<td>Test Report</td>
<td>5/18/2011</td>
<td>6/1/2011</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>5.8</td>
<td>Demo</td>
<td>6/1/2011</td>
<td>6/14/2011</td>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>
9. Libraries and Tools

**Eclipse**: It is an IDE that we are going to use for design. It is really compatible with Java programming language, and with the library provided by Minder to use BCID. Therefore we have chosen this IDE to the design the system.

**Emotiv BCID**: It is a brain computer interface device which gets the alpha, beta, theta etc. values with 50ms period (as default). Our sponsor company provides this device.
Emotiv Test Bench: It is a program to see data coming from the BCID. The program controls whether the points that should give alpha/beta values, is correctly placed to users head. Moreover, it concurrently shows the incoming values of data.

Java: Because of the libraries provided of use, we are going to use Java.

jFreeChart: It is a library to draw various types of graphs (line, pie chart etc.). It is available for Java programming language and distributed with GPL.

jMonkey: It is a game engine made especially for modern 3D development. The entire project is based on Java for high accessibility. Advanced graphics are supported through OpenGL 2 via LWJGL.

jOpenGL: It is similar to OpenGL in C/C++. It is extended from OpenGL for Java language. As mostly known, jOpenGL gives us a chance to design game environment.

swing: It’s a library to design frames, windows in given environment. In other words, it is a library to design GUI.

10. Conclusion

In conclusion, Detailed Design Report for Mindolog gives the definition, purpose and scope of the project. The possible design and other constraints that can be faced are explained. The tools and the libraries that will be used during developing the project are decided. Data flow models, class diagrams, interface features, entity relationship diagrams, possible use cases are given within the document. We have explained the works that we have done so far and within the schedule we give the future work to be done.