CENG492 Test Specifications Report

Buğra Şirin
Deniz Tav
Engin Fırat
Mert Özer

BiBER

April 24, 2011

Table of Contents

1. Introduction

1.1. Goals and Objectives

The purpose of this document is to define and outline the strategies and procedures for testing management for the final release of Mindolog. It covers the general methods made use of in the tests conducted for the project. Since Mindolog is a product for the psychologists and patients, and it is supposed to be used for several sessions one day at average by the end users, each unit and the whole system needs further considerations rather than just testing single functionalities just after they are developed. Quality assurance, verification and validation, reliability estimation, a generic metric, and trade-off between time and quality imply the importance of testing. The goal is to end up with a robust, bug-free and high-performance product after being finished all the testing process.

1.2. Statement of Scope

This document was prepared by the developer team of Mindolog, Biber. It covers descriptions of the methods used in the testing process of Mindolog, plan of the testing phase, procedures to be followed and techniques of keeping records during testing. During the development of the project Mindolog, each component of every module is supposed to be tested by the responsible developer(s) right after its implementation is finished. However, since many modules including many components work interactive in Mindolog, further separate testing of modules and testing of the whole system is absolutely necessary. To achieve this, we will elaborate on;

- What we will test
- What are our constraints while testing
- How will we handle severe, user-facing bugs
- How will we find internal bugs
- How our testing relates to our schedule
- How team members are responsible for different tests

1.3. Major Constraints

- **Time:** Mindolog is designed to perform several actions concurrently namely reading the data from the BCID device, showing it graphically and keeping it for further use while the user is playing the game. Furthermore, it can be run for different time intervals depending on the session. Therefore, optimization of all modules for better performance and any possible speed up is in the scope the time constraint.

- **Data:** Since Mindolog is composed of many modules running concurrently and passing data to each other during the run, minimizing the amount of data being transferred between modules is a goal in the sense of data constraint.

1.4. Definitions, Acronyms and Abbreviations

- BCID: Brain Computer Interface Device

1.5. References

- Pressman, Roger S. Software Engineering: A Practitioner's Approach, Sixth edition. New York, NY: McGraw-Hill
- IEEE Standard for Software Test Documentation

2. Test Plan

Before describing the overall testing strategy and the project management issues that are required to properly execute effective tests, it is important to emphasize some points that have effects on this process;

- *Time:* Our remaining 2 months to be spent for the project will be mostly occupied with implementation as it has been since the beginning of the spring semester. So we will do implementation and testing in parallel. Whenever a module is completed, or some modules are integrated, required unit tests and integration tests will be done immediately.

- *Usefulness of Obtained Test Results / Cost Ratio:* The ratio of *usefulness of test results/time* is the most important factor to determine whether a test is worth the time. Giving too much time to test some modules will be unnecessary when we consider the *test results/cost* since working on testing process requires both time and labor.

2.1. Software to Be Tested

We have 7 major components, namely: Core, BCID, chart, game, database, user interface and logger. Core module manages the rest of the modules because some of them run concurrently. Game module and chart module use the information coming from the BCID module, game module writes to the database and modules use logger module when necessary.

2.2. Testing Strategy

All of the components mentioned above will be tested thoroughly. Then, each integration will also be tested to ensure their interaction is as desired. Next, validation testing will be conducted to evaluate the component during the development process to determine whether it satisfies specified requirements. Lastly, system testing will be performed to ensure the robustness of the system as a whole.

2.2.1. Unit Testing

Core, BCID, chart, game, database, user interface and logger modules are needed to be tested. Since each module cannot be tested as individual parts for our project, other modules are going to be ignored during unit testing a module. The purpose of this phase is to ensure that each component does not have any internal errors.

2.2.2. Integration Testing

After all the modules pass the unit testing, they get to be tested whether they work correctly when they are running concurrently and communicating to each other. This specification is vital since although each module might be working individually, different implementations coming from different modules may conflict while running together which is to be located during integration testing and to be recorded for fixing.

Each of the other modules is connected to the Core module, so after the unit testing of each created module, it will be integrated to the core module and the integration of them will be tested. Apart from this testing, connection of other modules will be tested with the following procedure. Because there is a strong dependency between the Game and BCID module, their integration will be tested first. Second, the integration of chart module and game module will be tested because charts are drawn according to the game process. The third integration test process will be done with the integration of the database module. Because Game and Chart modules use the database, their interaction will be tested separately. Next, user interface will be integrated and tested considering the interaction with the game, chart and database modules. Lastly, the logger module will be integrated and tested based on the related modules.

2.2.3. Validation Testing

Result of this test is going to show whether expectations in design is met or not. Our Software Requirements Specification document defines the functional and non-functional requirements of the Mindolog Project. Every single requirement in this document must be considered one-by-one and the software must be guaranteed to satisfy all these requirements. Test cases and scenarios are defined and will be run to ensure the correct working of the project as a whole. The order of validation is the same as integration.

2.2.4. Higher-Order Testing

Various system tests are going to be held such as:

- **Performance Test:** Performance of the game is tested on different hardware for different options in the game.
- **Alpha/Beta Test:** The game is going to be tested by the children having attention deficiency and the results will be examined.

There will not be a stress testing since the input to the game is constant for each session and during the game.

There will not be a security testing for network because the system is going to run on a single machine.

2.3. Test Metrics

We will use the following metrics:

- Number Of Test Cases Executed
- Number Of Bugs Detected
- Number Of Bugs Fixed
- Number Of Priority Bugs Fixed

2.4. Testing Tools and Environment

There are no additional testing tools other than the game itself. Eclipse Debugger certainly will be used during the development and unit testing; however, different testing environments are required.

First environment type is hardware. While old hardware would be causing low fps, new hardware would be providing high fps. Thus, the game has to be tested with different hardware to see that its functions are hardware independent.

Second environment is *PgAdmin III*, which is a graphical user interface for the project's database management system PostgreSQL. We can see and check the database content after each insert, delete or update operation tests.

3. Test Procedure

In this chapter; strategies and procedures that are going to be followed in the test process are explained.

3.1. Unit Test Cases

Game: The game is the most need to be tested part of the project, because there are a lot of functionalities in the game. Our project game is a simple car race. We are supposed to test the left functionality, right functionality, and speed of the car. Besides, the game environment should be tested. For example, there should not be an end of the environment; the car should go unlimited on the road until the session till the end. Moreover, in the game module there is a test function which check the user is on the right strip, or not. The correctness of the strips is determined by the letters in a different panel during the game.

The function which checks the letters and the results of these checks must be tested whether their return values are right, or not.

Chart: The dynamic chart draws the brain wave values dynamically during a game playing session. We are able to change the values according to some thoughts and actions; moreover the values are recorded to the database concurrently. We will check the database and compare it with the dynamic graph with changing values. The other chart type is the one that can be seen anytime according to the past values. This graph also can be checked comparing the database values from *pgAdmin III*.

Database Connection and Querying: Information about patients, sessions and brain wave values are stored in the project database. Since additional database server is not going to be used, connection to database tests is not going to be performed. Instead of this, black box functionality tests are going to be performed such as insertion, deletion and update operations. After each test, results will be checked using *PgAdmin III* as explained in the section *Testing Tools and Environment*.

User Interface: In this project, static and dynamic graphical user interfaces are used. Features such as showing patient data and comparing them via graphs are static GUI elements while game play screen components such as the game itself and the dynamic graph are dynamic GUI elements.

Functionality test of this module is as follows:

- Does the correct sub menu open when proper button is pressed or clicked?
- Is the menu hierarchy complete?

Logger: According to the use cases that are mentioned in our Software Specifications Report, we will perform that cases and check the log whether it recorded right results or failed. We can find the failure point by tracing the records in the log file and comparing them with the steps of the cases.

3.2. Integration Testing

Because a communication exists between the modules through the Core module, it is essential to do integration testing while integrating modules to each other.
As soon as a new interaction or a new object is defined to be subscribed or published between modules, an integration test is applied to check correctness the communication. Therefore, this procedure lasts as long as the implementation continues.

3.3. Validation Testing

Validation testing is the testing procedure that is enabling compatibility between implementation and the other previous reports i.e. Software Requirement Analysis; Initial and Detailed Design Reports. Therefore, we divided this developer side testing procedure into two main parts that are Requirements Validation and Design Validation.

3.3.1. Requirements Validation

In requirements validation, satisfying functionalities that are claimed in requirement analysis is the general aim. So, to make this validation, our Requirement Analysis Report is traced, especially its Requirement Specification part. Requirement validation is made within the following subsections that are formed according to our Requirement Analysis Report.

- General Requirements: Where the menu will be displayed how to reach its items. To test these requirements, we will do what has been stated in requirements.
- Menu Items Requirements: What the items of the menu and their functions are. To test these requirements, we will check whether menu items accomplish their functionalities.

3.3.2. Design Validation

During the implementation, the most important consideration for us was to be consistent with the design. Moreover, generally our aim was to satisfy one to one correspondence between the class designs and implemented classes. Of course some changes have occured in the class design but there is nothing that is designed but not implemented, however there are some useful classes added to the design and implemented.

For design validation we are proposing to check the class and module interactions those are designed in the detailed design document. For this purpose we will use the white box testing method and verify&validate each interface function. By this way, we will make sure that our implementation is consistent with our design.

3.4. High-order testing

- **Performance Test:** We will test Mindolog on different machines to test the game part's performance. According to the fps rate, we will try to optimize the data transfer between the modules that run concurrently during the game playing session.
- **Alpha/Beta Test:** When we will release alpha version of our project, namely a game with a single level, we will test it with our group and close friend and see how it reacts to the measured brain waves during the session. Then we will fix any design problems or bugs, and after that we will release the beta version. Our main aim is to release beta version is to share it with the people from psychology area to get feedback. Thanks to this feedback we will be able to improve the features and fix functional defects in our system.

4. Testing Resources and Staffing

Each module will be tested firstly by its developer, so unit tests will be conducted by the developers of each module.

Integration testing will be conducted by the developers of the integrated modules on each integration phase.

Validation testing will be conducted by the developer(s) of the related module(s) with a member who is not a developer of that module for getting the benefit of outside perspectives.

System testing will be conducted by the whole team.

5. Test Work Products

The result of the testing process is the identification of the bugs. When a team member finds a bug, s/he will assign the correction to the developer of the corresponding module using TRAC.

6. Test Record Keeping and Test Log

For debugging purposes we plan to use Eclipse Debugger. The debugger includes necessary user interfaces; therefore we do not need any record keeping and logging.

7. Organization and Responsibilities

Main testing resource is the group members. Since group members are responsible for both developing and testing, there is a time constraint on testing. Thus, time spent on testing by group members has to be arranged in an efficient way. Although each developer has to test his/her own module during unit testing, another member should be included during the validation testing phase.

Staff responsibilities for testing different modules are seen below:

Module	Responsible Member(s)
Core	All team members
BCID	All team members
Game	All team members
Chart	Mert, Engin
User Interface	Deniz, Mert
Database	Buğra, Deniz
Logger	Engin, Buğra

Table 1: Staff Responsibilities

8. Test Schedule

Task	Dates
Unit Testing	May 2 – May 8
Integration Testing	May 9 – May 22
Validation Testing	May 23 – May 29
System Testing	May 30 – June 12

Table 2: Test Schedule