Software Test Plan
ClaritEar Mobile Application

educational and entertainment applications based on Android that are useful to train your hearing sensitivity in knowing the kind of tone or chord that is being played

May 17, 2016

Submitted By:
Muhammad Harist Refian
Pandu Wicaksono
Andira Rozawati

Table of Contents

1 Introduction

This document is a high-level overview defining our testing strategy for the ClaritEar Mobile Application. Its objective is to communicate project-wide quality standards and procedures. It portrays a snapshot of the project as of the end of the planning phase. This document will address the different standards that will apply to the unit, integration and system testing of the specified application. We will utilize testing criteria under the white box, black box, and system-testing paradigm. This paradigm will include, but is not limited to, the testing criteria, methods, and test cases of the overall design. Throughout the testing process we will be applying the test documentation specifications described in the IEEE Standard 829-1983 for Software Test Documentation.

1.1 Team Interaction

The following describes the level of team interaction necessary to have a successful product.

- The Test Team is responsible for visualizing test cases and raising quality issues and concerns during meetings to address issues early enough in the development cycle.

- If an area is not acceptable for testing, the code complete date will be pushed out, giving the developers additional time to stabilize the area.

- Since the application interacts with a back-end system component, the Test Team will need to include a plan for integration testing. Integration testing must be executed successfully prior to system testing.

2 Test Objective

The objective our test plan is to find and report as many bugs as possible to improve the integrity of our program. Although exhaustive testing is not possible, we will exercise a broad range of tests to achieve our goal. We will be testing a ClaritEar Application. There will be eight key functions used to mange our application: load, store, clear, search, insert, delete, list in ascending order, and list in descending order. The application will only be used as a demonstration tool, but we would like to ensure that it could be run from a variety of platforms with little impact on performance or usability.

3 Process Overview

The following represents the overall flow of the testing process:

1. Identify the requirements to be tested. All test cases shall be derived using the current Program Specification.

2. Identify which particular test(s) will be used to test each module.

3. Review the test data and test cases to ensure that the unit has been thoroughly verified and that the test data and test cases are adequate to verify proper operation of the unit.

4. Identify the expected results for each test.

5. Document the test case configuration, test data, and expected results.

6. Perform the test(s).

7. Document the test data, test cases, and test configuration used during the testing process. This information shall be submitted via the Unit/System Test Report (STR).

8. Successful unit testing is required before the unit is eligible for component integration/system testing.

9. Unsuccessful testing requires a Bug Report Form to be generated. This document shall describe the test case, the problem encountered, its possible cause, and the sequence of events that led to the problem. It shall be used as a basis for later technical analysis.

10. Test documents and reports shall be submitted. Any specifications to be reviewed, revised, or updated shall be handled immediately.

4 Testing Process

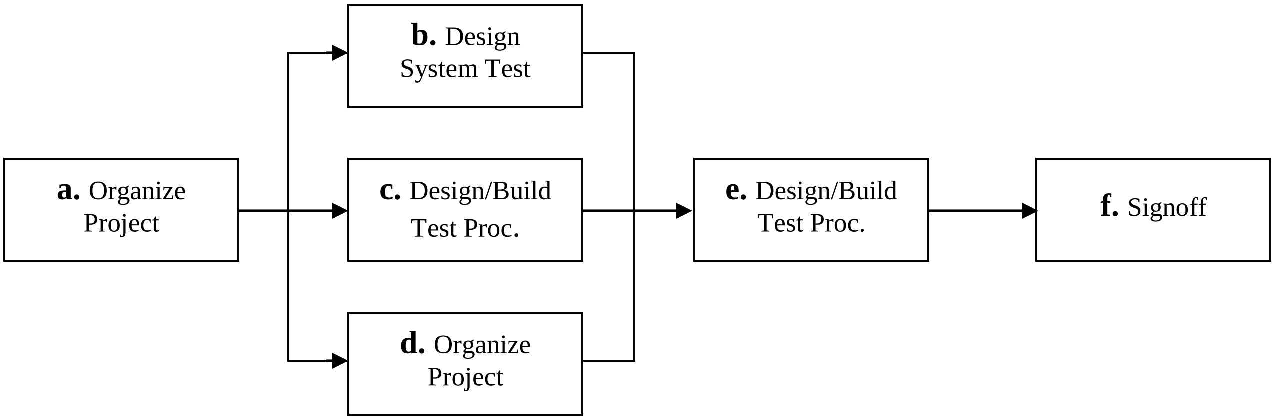

Figure 1: Test Process Flow

The diagram above outlines the Test Process approach that will be followed.

a. **Organize Project** involves creating a System Test Plan, Schedule & Test Approach, and assigning responsibilities.
b. **Design/Build System Test** involves identifying Test Cycles, Test Cases, Entrance & Exit Criteria, Expected Results, etc. In general, test conditions/expected results will be identified by the Test Team in conjunction with the Development Team. The Test Team

will then identify Test Cases and the Data required. The Test conditions are derived from the Program Specifications Document.

c. **Design/Build Test Procedures** includes setting up procedures such as Error Management systems and Status reporting.

d. **Build Test Environment** includes requesting/building hardware, software and data set-ups.

e. **Execute System Tests –** The tests identified in the Design/Build Test Procedures will be executed. All results will be documented and Bug Report Forms filled out and given to the Development Team as necessary.

f. **Signoff** - Signoff happens when all pre-defined exit criteria have been achieved.

5 Testing Strategy

The following outlines the types of testing that will be done for unit, integration, and system testing. While it includes what will be tested, the specific use cases that determine how the testing is done will be detailed in the Test Design Document. The template that will be used for designing use cases is shown in Figure 2.

Tested By:	
Test Type	
Test Case Number	
Test Case Name	
Test Case Description	
Item(s) to be tested	
1	
2	
Specifications	
Input	**Expected Output/Result**
Procedural Steps	
1	
2	
3	
4	
5	
6	
7	

Figure 2: Test Case Template

5.1 Unit Testing

Unit Testing is done at the source or code level for language-specific programming errors such as bad syntax, logic errors, or to test particular functions or code modules. The unit test cases shall be designed to test the validity of the programs correctness.

5.1.1 White Box Testing

In white box testing, the UI is bypassed. Inputs and outputs are tested directly at the code level and the results are compared against specifications. This form of testing ignores the function of the program under test and will focus only on its code and the structure of that code. Test case designers shall generate cases that not only cause each condition to take on all possible values at least once, but that cause each such condition to be executed at least once. To ensure this happens, we will be applying Branch Testing. Because the functionality of the program is relatively simple, this method will be feasible to apply.

5.1.2 Black Box Testing

Black box testing typically involves running through every possible input to verify that it results in the right outputs using the software as an end-user would. We have decided to perform Equivalence Partitioning and Boundary Value Analysis testing on our application.

5.1.2.1 Equivalence Partitioning

In considering the inputs for our equivalence testing, the following types will be used:

- Legal input values – Test values within boundaries of the specification equivalence classes. This shall be input data the program expects and is programmed to transform into usable values.
- Illegal input values – Test equivalence classes outside the boundaries of the specification. This shall be input data the program may be presented, but that will not produce any meaningful output.

The equivalence partitioning technique is a test case selection technique in which the test designer examines the input space defined for the unit under test and seeks to find sets of input that are, or should be, processed identically. The following table represents our equivalence classes, both valid and invalid.

Input/Output Event	Valid Equivalence Classes	Invalid Equivalence Classes
Input maximum number of allowed values	25 values	> 25 values
Input biginteger	bigintegers between -9223372036854775808 and 9223372036854775808	bigInteger > 9223372036854775808 bigInteger < -9223372036854775808 Non-integers (characters) Non-integers (decimal values)
Load external file	Comma delimited file with only one value per line File exists	No commas Multiple entries per line No file content File does not exist
Store external file	File exists	File does not exist

5.1.2.2 Boundary Value Testing

The acceptable range of values for this application was set by the development team. Due to the limitations of the GUI, the developers also limited the size of the input values to three digit integers. The valid and invalid ranges are shown below along with the corresponding valid and invalid boundary test values.

Acceptable Range: -9223372036854775808 ≤ x ≤ 9223372036854775808
Invalid Range: -∞ < x < -9223372036854775808 and 9223372036854775808 < x < + ∞

Valid Boundary Tests:

Boundary$_1$: x = -9223372036854775808
Boundary$_2$: x = 0
Boundary$_3$: x = 9223372036854775808

Invalid Boundary Tests:

Boundary$_4$: x = 1000000000000000000000
Boundary$_5$: x = -100000000000000000000
Boundary$_6$: x = -9999999999999999999

Boundary$_7$: x = 9999999999999999999

5.2 Integration Testing

5.2.1 Incremental Testing

There are two primary modules that will need to be integrated: the Graphic User Interface module and the Shop Repository module (back-end). The two components, once integrated, will form the complete ClaritEar Mobile Application. The following describes these modules as well as the steps that will need to be taken to achieve complete integration. We will be employing an incremental testing strategy to complete the integration.

Module 1 - Graphic User Interface (GUI) Module

This module provides a simple GUI where the user can perform the different actions (functions). This module will be tested separate from the backend to check if each interface (e.g. insert button) is functioning properly, and in general, to test if the mouse-event actions are working properly. The testing will be performed by writing a stub for each element in the interface.

Module 2 – ClaritEar MobileBackend Module

This module will be tested separate from the GUI by printing out the results to the Console. In testing this module we will follow the incremental testing method i.e. testing one function first and then keep adding additional function and test it again until all the required functions are tested.

When the GUI is combined with the backend module, we will have a complete ClaritEar Mobile application. To achieve complete integration of these two modules, we will test each element in the GUI by replacing the stubs with the appropriate function from the back end. The results will be displayed within the GUI instead of through the Console. In testing the combined modules, we will follow the incremental testing method. Each stub will be replaced one at a time and tested. This will be done until all stubs have been replaced by the appropriate functions from the backend.

5.3 System Testing

The goals of system testing are to detect faults that can only be exposed by testing the entire integrated system or some major part of it. Generally, system testing is mainly concerned with areas such as performance, security, validation, load/stress, and configuration sensitivity. But in our case well focus only on function validation and performance. And in both cases we will use the black-box method of testing.

5.3.1 Function Validation Testing

The integrated “ClaritEar Mobile Application” will be tested based on the requirements to ensure that we built the right application. In doing this test, we will try to find the errors in the inputs and outputs

In addition, we will test:

- The interfaces to ensure they are functioning as desired (i.e. check if each interface is behaving as expected, specifically verifying the appropriate action is associated with each mouse_click event).
- The interaction between the GUI and the backend repository. In this case the data will be inserted and check if they are processed in the backend and give the expected output.

5.3.2 Performance testing

This test will be conducted to evaluate the fulfillment of a system with specified performance requirements. It will be done using black-box testing method. And this will be performed by:

- Storing the maximum data in the file and trying to insert, and observe how the application will perform when it is out of boundary.
- Deleting data and check if it follows the right sorting algorithm to sort the resulting data or output.
- Trying to store new data and check if it over writes the existing once.
- Trying to load the data while they are already loaded

6 Entry and Exit Criteria

This section describes the general criteria by which testing commences, temporarily stopped, resumed and completed within each testing phase. Different features/components may have slight variation of their criteria, in which case, those should be mentioned in the feature test plan. The testing phase also maps to the impact level definition when a defect is entered in the bug-tracking phase.

6.1 Unit Testing

Unit Testing is done at the source or code level for language-specific programming errors such as bad syntax, logic errors, or to test particular functions or code modules. The unit test cases shall be designed to test the validity of the programs correctness.

6.1.1 Black Box Phase

Black box testing typically involves running through every possible input to verify that it results in the right outputs using the software as an end-user would. We will use Equivalence Partitioning and Boundary Value Analysis complexity metrics in order to quantifiably determine how many test cases needed to achieve maximum code coverage.

6.1.1.1 Black Box Entry Criteria

The Black Box Entry Criteria will rely on the component specification, and user interface requirements. Things that must be done on entry to the Black Box stage:

- All ClaritEar Mobile functions must either be coded or stubs created.
- The type of Black Box testing Methods will be determined upon entry. We will use Equivalency Partition, and Boundary Value Analysis.
- Equivalency Partition will include, Integer data types only, No Character data types accepted, each data field will be comma delimited, and there will be 1 value per line in the data file.

6.1.1.2 Black Box Exit Criteria

The Black Box Exit Criteria listed below explains what needs to be completed in-order to exit Black Box phase. To exit the Black Box phase 100% success rate must be achieved. Things that must be done upon exiting the Black Box stage:

- The Equivalence Classes will have been created for the valid and invalid input values. For our ClaritEar Mobile the input domain values for Equivalence Partitions will include Integer data types only, each data field will be delimited by a comma and carriage return, and one data value per line in the input data file.
- The invalid input domain values for Equivalence classes will include loading an empty input data file, entering a delimiter other that a comma, and entering more than one data value per line in the input data file.
- Boundary Value Analysis will have generated Test Cases based on the boundary values of Integer data type values. These Test Cases will test for values above and below the specified boundary values. For example, values that include infinity, negative infinity, zero, and decimal numbers.
- All code bugs that are exposed are corrected.

6.1.2 White Box Phase

The White Box criteria apply for purposes of focusing on internal program structure, and discover all internal program errors. Defects will be categorized and the quality of the product will be assessed.

6.1.2.1 White Box Entry Criteria

The White Box Entry Criteria will rely on the QA engineers verifying that the major features work alone but not necessarily in combination; exception handling will not be implemented. The design and human interface are stable. Things that must be done on entry to the White Box stage:

- All ClaritEar Mobile functions must be coded.
- Black Box Testing should be in its late stages.

After the White Box criteria have been met, the product enters the White Box stage. During White Box stage Development Engineering’s emphasis is on refining the product and fixing defects. Information Design’s emphasis is on developing product user documentation.

6.1.2.2 White Box Exit Criteria

ClaritEar Mobile Apps in the White Box stage should have a generally stable feel to it. White Box testing continues until the Black Box or next milestone criteria are met. To exit the White Box phase 100% success rate must be achieved. The following describes the state of the product upon exit from the White Box Stage:

- All ClaritEar Mobile functions are implemented, operational and tested.
- All Branch Testing test cases will be generated. The test cases will be generated from the Control Flow diagrams of all functions.
- All code bugs that are exposed are corrected.

6.2 Integration Test

There are two modules that will be integrated for Integration Testing. The two modules are The Graphic User Interface module and database(back-end). The two components will consist of a mixture of stubs, driver, and full function code. The following describes the entry and exit criteria for Integration testing.

6.2.1 Integration Test Entry Criteria

The Integration Test Entry Criteria will rely on both modules to be operational. Things that must be done on entry to the Integration Test stage:

- All ClaritEar Mobile function must either be coded and/or stubs created.
- The Graphical User Interface must either be coded and/or a driver and stubs must be created. The driver is implemented to facilitate test case input and output values.
- Interfaces and interactions between the Shop Module and the Graphical User Interface must be operational.
- Black Box Testing should either be in its late stages or completed.
- White Box Testing should have begun.

6.2.2 Integration Test Exit Criteria

The Integration Test Exit Criteria will rely on both modules to be operational. The ClaritEar Mobile design and human interface must be stable. To exit the Integration Testing phase 100% success rate must be achieved. Things that must be done on exit from the Integration Test stage:

- All code bugs that are exposed are corrected.
- The Shop Module and Graphical User Interface Module will interact together with complete accuracy, according to the System Specification Design. All discrepancies are corrected.

- Both Modules are ready for System Testing. Stubs and drivers are replaced with fully functional code.
- Black Box Testing is completed.
- White Box Testing should either be in its late stages or completed.

6.3 System Test

The System Test criteria apply for purposes of categorizing defects and the assessing the quality level of the product. All elements of the Shop Module and Graphical User Interface are meshed together and tested as a whole. System test focuses on functions and performance, reliability, instillation, behavior during special conditions, and stress testing.

6.3.1 System Test Entry Criteria

The Entrance Criteria specified by the Development Engineers, should be fulfilled before System Test can commence. In the event, that any criterion has not been achieved, the System Test may commence if both Development and Test Engineers are in full agreement that the risk is manageable.

- The Graphical User Interface and the ClaritEar Mobile back-end Module must be fully functional.
- All developed code must be unit tested. Unit and Link Testing must be completed and signed off by the development team.
- All test hardware and environments must be in place, and free for System test use.
- All Black Box testing must be complete and exposed bugs must be corrected.
- All White Box testing must be complete and exposed bugs must be corrected.
- Integration Testing must be complete and exposed bugs must be corrected
- The Graphical User Interface will be the method of interacting with the system, so the GUI will be tested thoroughly.
- Development and Test Engineers agree that Function Validation Testing will cover function performance, reliability, stress and load testing.

6.3.2 System Exit Criteria

The Exit Criteria must satisfy all the criteria listed below. This verifies that all elements of the project mesh properly. This is to make sure that all the system functions and performs according to the System Specification Document.

- All Function Validation Testing is 100 percent successful. Testing for all ClaritEar Mobile functions interact with complete accuracy.
- No degradation of System performance across different platforms of Windows operating system will be affected. (Windows 7 or above is acceptable)
- The Graphical User Interface performs to System Specification Requirements.
- All the Shop properties are expressed correctly through the Graphical User Interface.
- All input fields on the Graphical User Interface are working correctly.

- All high priority errors from System Testing must be fixed and tested.
- If any medium or low-priority errors are outstanding – the Development Engineers and Test manager must sign off the implementation risk as acceptable.

7 Bug Tracking/ Bug Process

During testing, the testing team members normally encounter behavior that goes against a specified or implied design requirement in the product. When this happens, we will document and reproduce the bugs for the developers.

Expectation of a bug:

- Keep track of what version of the application the bug is found
- Determine if bug has already been written up
- Indicate the steps to reproduce the bug – write enough details for others looking at the bug to be able to duplicate it; exclude unnecessary steps (i.e. If access point is irrelevant, be more general in your steps).
- Actual results – be specific on your findings.
- Expected results – how the product should behave based on the specified or implied requirements.
- Implications – How does the defect affect the quality of the product?

The following chart defines the impact levels to be used when entering bugs.

Impact	Definitions
1 – Fatal	**Test Stopper:** If you can’t access a function and need the bug to be fixed immediately. The defect prevents QA from testing the feature area, sub-area or functionality of the feature.
2 – Serious	**Beta Stopper:** This is a bug that users would experience such as: data corruption, calculation errors, incorrect data, UE’s and system crash on common user scenarios, significant QA risk, and major UI defects.
3 – Minor	**Live Release:** A bug that must be fixed before the product is officially completed, UE’s or crashes, content, and UI and graphic changes required for release.

7.1 Various Roles in Bug Resolution

- **Author –** The person who wrote the bug; this will be someone on the QA team
- **Resolver –** Normally an Engineer assigned to a specific area of the application.
- **Verifier –** normally a QA Engineer responsible for testing the fix and closing the bug.

7.2 Bug Report Form

VHTN Software Development

Problem Report #: ________

Program ______________________ Release ______________ Version ________

Report Type (1-6)_____
1 - Coding error
2 - Design issue
3 - Suggestion
4 - Documentation
5 - Hardware
6 - Query

Severity(1-3)_____
1 - Fatal
2 - Serious
3 - Minor

Attachments (Y/N) ________
If yes, describe:
____
____

Problem Summary ____

Can you reproduce the problem? (Y/N) ________

Problem & how can it be reproduced?

____
____
____

Suggested fix (optional input)

____
____
____

Reported By: ______________________________ Date: ______________

Items below are for use only by the development team

Functional Area:_____________________ Assigned To:_____________________

Comments:

____
____

Status_____
1 - open *2 - closed*

Prioirity(1-5)_____

Resolution(1-9)_____ Resolution Version ________

1 - Pending *4 - Deferred* *7 - Withdrawn by reporter*
2 - Fixed *5 - As designed* *8 - Need more info*
3 - Irreproducible *6 - Can't be fixed* *9 - Disagree with suggestion*

Resolved By: ______________________________ Date: ______________

Tested By: ______________________________ Date: ______________

Treat as Deferred (Y/N)_____

8 Roles and Responsibilities

8.1 Development Team

Code Development Project Leader – Pandu Wicaksono

- Ensure Phase 1 is delivered to schedule and quality
- Ensure exit criteria are achieved prior to system test signoff
- Regularly review testing progress with test controller.
- Raise and manage issues/risks relating to project or outside test teams control.
- Review and sign off test approach, plans and schedule.

8.2 Testing Team

Test Planner / Controller – Muhammad Harist Refian

- Ensure Phase 1 is delivered to schedule and quality
- Produce high level and detailed test conditions
- Produce expected results
- Report progress at regular status reporting meetings
- Co-ordinate review and signoff of test conditions
- Manage individual test cycles and resolve tester queries/problems.

Lead Tester – Andira Rozawati

- Identify test data
- Execute test conditions and mark-off results
- Prepare software error reports
- Administrate error measurement system
- Ensure test systems outages/problems are reported immediately and followed up.
- Ensure entrance criteria are achieved prior to system test start.
- Ensure exit criteria are achieved prior to system test signoff.

9 Test Schedule

The section contains the overall project schedule. It discusses the phases and key milestones as they relate to quality assurance. It discusses the testing goals and standards that we'd like to achieve for each phase of testing that will be deployed, e.g., Usability Testing, Code Complete Acceptance, Beta Testing, Integration Testing, Regression Testing, System Testing.
The key dates for overall ClaritEar Mobile development and Testing are outlined below. For details on the schedule, refer to SPMP. For details on general Engineering QA deliverables, refer to the test plan document.

Milestones	End Date	Notes	QA Deliverables/Roles
Planning Phase	03/03/16	At this Milestone, the high level	High-level test planning activities,

Milestones	End Date	Notes	QA Deliverables/Roles
		planning should be completed. Some of the deliverables are: Project Plan, Program function specifications.	which include preliminary development of Master QA Plan (this document, QA schedule.
Design Phase	29/03/16	This is a feature-driven milestone where the requirements and initiatives are further defined and solutions are finalized. The deliverables for this phase are Program source code and other design related documents.	Development and Test engineers participate actively in feature design by inspecting and reviewing the requirements and design documents. As the design documents are completed, the test engineers are encouraged to start working on the Test Plan document and test design planning.
Code Complete -Infrastructure	30/04/16	This milestone is when all infrastructure development and functions should be complete. The testing team should have preformed unit & integration testing before checking the code into any build.	The Test Engineers should have completed or in the final stages of their preliminary Infrastructure Test Plan, test cases and other QA documents related to test execution for each feature or component such as test scenarios, expected results, data sets, test procedures, scripts and applicable testing tools.
Code Complete -Function	22/05/16	This milestone includes unit testing and code review of each function component prior to checking the code into the test phase. The deliverables include system-testing specification, Unit testing specifications, Integration plan.	The Test Engineers should have provided Code Complete Assessment Test to Development Engineer one week prior to Code Complete Review date. The Test Engineers should also have completed or in the final stages of their preliminary White Box Test Plan, test cases and other QA documents related to test execution for each feature or component such as test scenarios, expected results, data sets, test procedures, scripts and applicable testing tools.
Beta Ready	31/06/16	This milestone represents that all features are ready for Beta release shutdown.	2 Weeks regression of ClaritEar Mobile features to Beta and preparation for Beta Shutdown.
Ship/Live	07/06/16	Product is out.	Any unfinished Testing documents should be complete.

10 Quality survey

10.1 The form

	Questions	Completely disagree	Disagree	Neutral	Agree	Completely agree
1	You can easily choose things in ClaritEar Mobile					
2	The app is recognizing your exact walk and direction					
3	The user interface is user-friendly					
4	The app does not crash too frequently					
5	You can check out and buy things in ClaritEar Mobile					

The tester will answer the following questions by choosing between the five (5) possibilities. These questions are about the tester's experience about Tomorrow-Shop, and should be answered honestly.

11 Deliverables

- Program function specifications

- Program source code

- Test plan document - this document should address testing objectives, criteria, standards, schedule and assignments, and testing tools.
  - Unit Testing Plan
  - Integration Plan
  - System Testing Plan

- Test Design Document
  - Unit white-box test design – covers white testing criteria, methods and test cases
  - Unit black-box test design – covers black-box testing criteria, methods and test cases
  - System test design – covers system test criteria, methods, and test cases, scripts.

- Test report document
  - Unit white-box test report – covers unit white box test results, problems, summary and analysis

- Unit black-box test report – covers unit black box test results, problems, summary and analysis
- System Test report – covers system test results, problems, summary and analysis

12 References

Pressman, Roger S. Software Engineering - A Practitioner's Approach. Fifth edition. The McGraw-Hill companies, Inc.

Kaner, C., Falk, J., Nguyen, H.-Q. Testing Computer Software. Wiley Computer Publishing, 1999.