Acceptance Criteria

Software Engineering Group 6

0/3/2012: Acceptance Criteria, v2.0

March 2012 - Second Deliverable

Contents: **Page no:**

Introduction

This is the Acceptance Criteria document deliverable that will contain all of the proposed tests that will be carried out and implemented into the software development of the Software Engineering Project. The reason for why an Acceptance Criteria is required shall be justified within this document which will account for the two main parts of the deliverable: Test Environment and the Acceptance Tests. There will also be sub sections that will hold further information about the types of testing the Quality Assurance Team would like to conduct if deemed necessary and relevant. Therefore the classifications of types of testing will be established ensuring that not only the program software is tested but also the constituent parts that may affect the design and development of the product.

The Acceptance Criteria will be satisfying the specifications stated by the Analysis Team's Analysis Model so that an outline and guidance for the appropriate testing may be formalised. So through the use of the Analysis Model as well as the Requirements Specification the Acceptance Criteria will only have the critical specific variables that are required to conduct the testing processes for the software development.

The Test Environment will contain the anticipated and target system requirements that the machines that the software will want to run on. This will also state further information on primary and secondary file (data) packages that may be required to allow testing on the software. Moreover, the Test Acceptance Test shall incorporate the relevance of Black Box Testing and White Box Testing where it is appropriate on the series of proposed tests within this document.

Test Environment

As the Acceptance Criteria implements specifications from the deliverable Requirements Specification, the information required on what system requirements are necessary for the software to run can be stated:

Operating System: Windows 7

Processor: AMD Athlon(tm) 64 X2 Dual Core Processor 3800+ 2.00 Ghz

OR Intel® Core ™ EQUIVALENT

RAM: 2 GB DDR 2

System Type: 64 - bit Operating System

Software Required: Web browser, Mozilla Firefox v10.0.1

Data Files Required: Ant Game Source Code & Ant Brains

Load Conditions:

Graphics Processing Unit: ATI Radeon HD 5450 (Minimum)

Testing Software Required: Java Testing Software - JUnit, JavaScript Testing - QUnit

Throughout the testing process and by going through each of the proposed tests found in the Acceptance Tests section, all of the tests will be conducted accordingly and for a robust process of testing, the Quality Assurance Team will ensure to devise a recording/archiving system that will store and monitor the results of the tests applied to the different contexts, not only to the designed software but that may which also effect the Users of the program.

For definite quality testing and unit proofing the source code, the Quality Assurance Team will try to conduct tests at every major milestone of the development of the source code. This will also make sure that early detection of bugs and possibly modifications may be implemented and deployed quickly for the better software engineering as a whole.

By having an archiving/recording system, the Quality Assurance Team will be able to check through the results fortifying that the tests and findings within the content abide by the specifications stated not only within the Requirements Specification but possibly through the Project Specification and Quality Manuals too. This will ensure a fair control is present within the entire testing of the project software.

There will be the required Ant Brains that will fall into the category of Data Files Required. As well as that the most important data file will be the source code itself. These two data files will compliment one another in the running of the overall Ant Game.

Acceptance Tests

This is structured according to the Requirements Specification, for each test, following information shall be provided: section of the Requirements Specification being tested, Prerequisites (data files) for running the test, test to be performed and Expected Result(s).

Acceptance Tests are black box tests. White box tests are specified in the Test Specification.

Test No.	Test Description	Prerequisite(s)	Action(s)	Expected Result(s)
0.1	GUI Testing	Application itself	Inspect the application by playing it	The graphics, buttons, interface of the game should be displayed clearly for the users
0.2	GUI Functionality Testing	Application itself	Inspect the application by playing it	The buttons that the users click must respond in a logical result. E.g. Eat = eat the food
0.3	Smoke Testing (Basic testing)	Application itself	Run the application	The application should run smoothly with no errors. All the functionalities of the application should be in order.
0.4	None Functional Testing	Application itself	Enter the unexpected, invalid inputs for the application and see how it responds	The application tolerates the inputs, manages the error by applying the error-management routines such as displaying an error messages to tell the user to provide the right inputs
0.5	Load Testing	Application itself	Run the application	The loading time should not take longer than 5 seconds in any executions of the application
0.6	Stress Testing	Application itself	Run the application in an unexpected condition, e.g. a lot of food or users	The application should still be able to run

0.7	Usability Testing	Application itself	Run the application	The users should be able to understand what is going on in the application, should be able to navigate to anywhere they desire
0.8	Sound Feature Testing	Application itself	Run the application	The sound should output on the available audio devices at the correct time of execution
0.9	Smoke Testing	Application itself	Run the application	The scores of the players should start from 0 at the beginning of the game
1.0	In-game Testing	Application itself	Run the application and play the game for a while	The correct score of each player should output to the GUI
1.1	End of game Testing	Application itself	Run the application and play the game until it finishes	The end of the game should result in the application providing a neat leader board screen
1.2	Accessibility Testing	Application itself	Run the application	The application should be controlled easily with the keyboard or allocated peripherals available
1.3	Bottom-up Testing	Application itself	Run the application from the beginning	The application will run from the start, the scores of the players should slowly increase depending on the scores they have and the game will finish when the food is finished or the enemy team(s) have been eliminated
1.4	Benchmark Testing	Application itself and a free benchmark testing application	Run the application and the benchmark testing application	The benchmark testing application will analyse the application as it's running with it, results will be shown at the end of the test

1.5	Compatibility Testing	Application itself and a few other devices with different operating systems	Run the application on different devices with varying system specifications	The application should be able to run smoothly on the computers with any operating systems that support Java and its platforms (JavaScript)
1.6	Branch Testing	Application itself (complete version)	We will test the application from beginning to end, we'll make sure every single line of code from the application is being used	There shouldn't be errors, should be running smoothly and no irrelevant or unused code present
1.7	Depth Testing	Application itself	Test the application, playing with all the features it has	All the features should be functioning well
1.8	Gorilla Testing	Application itself	Break down the application, test each particular module one at a time, we will make sure that we test it to its limit	It should run smoothly if the code has been made sure that it has error management protocols and routines
1.9	Installation Testing	Application itself and our test machine	Install the game in our test machine	The game should be able to install in our machine and run smoothly
2.0	Ant brain text file	Ant brain text file	Make sure the structure of the ant brain text file is correct	Ant brain will be accepted if the structure is right. E.g. each line describes no more than one state

Types of Testing

For this section of the Acceptance Criteria, all of the classes of testing that will and may be used will be documented and described to give an in depth account of the details that could possibly be used on the different assets that surround the project as a whole but most importantly, for the testing of the JavaScript, software product.

Brief descriptions of these tests were given previously within the Project Plan, the first deliverable for the Software Engineering project for the customer. Within it, it was stated that these tests must be conducted to allow for the correct and smooth execution of the program. As well as that, the tests will make sure that the specifications and what the customer wants are met and satisfied to the very best of what was stated within the Requirements Specification.

Black Box and White Box Testing will perhaps be the two most important types of testing that will be used throughout and therefore an in depth outlook within those two categories will be given. Moreover, Smoke Testing and User Acceptance Testing can be found within this section that will also have information and the justifications as to why they have been included as well as the relevancy included.

By conducting these tests, the Quality Assurance Team can assure the Programming Team and the entire Group - as a whole - that the software is bug free, the customer will be satisfied and all specifications have been critically met.

In the following sections, the types of testing will now be introduced outlining their relevance and probable importance if circumstances do arise that they should be used.

Smoke Testing

Smoke testing in the software development/engineering context, is the first kind of testing that is conducted before further testing is implemented. Through the application of smoke testing, the Quality Assurance Team should be able to detect the obvious errors and bugs that could possibly be present within the source code and other deliverable documents that effect project as a whole.

This kind of test focuses on the vital functionalities and probable design such as the User Interface of the software as only the important aspects of the software can be refined. A basic example of smoke testing can be demonstrated in this question: "Does the program take in other constituent data files such as the Ant Brains?". This will therefore assure the Quality Assurance Team of severe problems if they do exist as a non-executable program cannot be put forward for delivery to the Customer as a critical error persists.

'Smoke testing performed on a particular build is also known as a build verification test.' (Software Development, Wikipedia. 2012.)

The above quotation emphasises the vital principal of smoke testing, it is a verification test of all basic elements that make software a complete running product that should not have any errors and can be delivered to the Customer.

Not only does smoke testing allow for the validation of key concepts it also verifies that all code within the software are operational.

Black Box Testing

Black box testing can be called as functionality testing. It is to test if the function of the application is working or not. During the test, you will see the application as a box that cannot be opened, which means that you will not consider what is really happening inside the application such as internal structure and application's code. Putting the inputs to the application and determine the outputs according to the requirements from the specification, to see if the application is able to receive appropriate input data and also able to produce the correct output data. This black box testing focus on the external structure of the application, this is a main test for the interface and features of the application.

Commentary

Black box testing is based on the user/ customer's point of view to test the correspondence between the inputs and outputs. Hence, black box testing is not a test that would be able to tell if the application has design problems.

Function

Black box testing has a role that mainly finds the following types of errors:

- Interface error
- Performance error
- Data base error
- Function incorrect or missing
- Initialization error
- Termination error

Outline

Black box testing has to test with all the possible inputs in order to detect the errors correctly from the application. There are numbers of tests in black box testing, we have to test with the valid inputs and also the illegitimate inputs. However, we have to conduct targeted testing in order to get the application fully tested.

Black box main testing test case methods are listed below:

- Equivalence partitioning
- Boundary value analysis
- Error guessing

White Box Testing

White box testing is based on structural testing or logic- driven testing. It is to test the procedures of the application depending on the internal structure of the application. We could detect the internal action of the application is in accordance with the provisions of the design specification. We could detect if each platform or pathway is determined to function well or not. This method is to see the application like a box which can be opened, which means that the testers (QA team) do need to understand what is happening internally of the application such as the logical structure of the code and the design of the code. The testers (QA team) will test all the logic paths through the state inspection at different points to see if the actual state is consistent with the expected state.

Outline

There are two categories of testing methods, static and dynamic testing methods. Static testing method is based on using the simulation technology to run the test and analysis to the application. Dynamic testing method is to enter a set of preset data which built according to the specification to test the application until it finds the errors.

White box testing main testing methods are listed below:

- Code checks
- Static structural analysis
- Domain test
- Basic path testing

User Acceptance Testing

User Acceptance Testing may only take effect once an agreement of specific requirements and objective goals have been determined and identified. This information can be gathered from the Requirements Specification whom the Analysis Team is responsible for as well as (if personally stated) the Customer himself has specified and requested for compulsory requirements.

It is standard practice that conducting this test will require volunteers from the target users that understand the Customer's criteria and can therefore provide a judgement whether the software engineering team was able to fulfill all of the demands that were previously set beforehand. In doing so, if the User Acceptance Testing is 100% effective in its success and purpose, the new developed system will be more refined in terms of a full proofed product now possible to submit and deliver to the final Customer.

An example of a basic User Acceptance Test would be the relationship between the user and the program itself. Take for example the interaction between the user and the program's interface, if the interface works correctly where the game is accessible within a click or few, the interface's functionality can be deemed a success. On the other hand, in terms of user interface aesthetics, we can ask the question:

'Does the user interface for this program appeal and entice you to play the game?'

If so, we can also conclude that the user interface on the context of User Acceptance Testing can be accepted as a fully successful element within the software.

When the time comes that a User Acceptance Test must be conducted upon the apparent final software, usually, this kind of test can be classed a final verification of all systems that have been designed and implemented within the Ant Game. This test shall focus on the functionality, aesthetics, relevancy and most importantly the Users' needs that should be present within the design of the game.

Specific implementation of User Acceptance Testing (Wikipedia 2012), shows all of the different approaches and variables that must be Incorporated to be able to conduct a successful class of this kind of testing.

Conclusion

Within this document, an outlook of all of the types of testing and the proposed purposes and justifications of such tests have been provided and described. More importantly, the Test Environment(s) and the Acceptance Tests have been established and identified. As these two main elements have been clarified, a foundation for the acceptance of what the team is to produce is now official in the application of testing all created aspects that will affect the source code and the project itself.

By proposing a foundation for testing, the software engineering project for the target Ant Game may be approached not only systematically but also efficiently and effectively as the testing will allow for the assurance of well-designed software and a level of customer satisfaction met.

The most important thing that should be considered in the application of these tests will be that by allowing for these tests to be conducted, the team may be able to look at the software how it would perform once it is in production and available in the market and especially to the target users.

As a whole, through the incorporation of these tests and their practice implementations, assurance that the Customer will have a fully functional and enticing product that successfully deploys the initial system proposed at the beginning of the project and its development stages. All of the test should cater for every constituent element(s) that affect the project and the software itself as a complete software deliverable.

Bibliography

1. Software Development, 2012. Wikipedia. [online] Available at: <http://en.wikipedia.org/wiki/Smoke_testing#Software_development> [Accessed 6/3/2012].
2. User Acceptance Testing, 2012. Wikipedia. [online] Available at: <http://en.wikipedia.org/wiki/Acceptance_testing> [Accessed 6/3/2012].
3. Black-box testing, 2012. Wikipedia. [online] Available at <http://en.wikipedia.org/wiki/Black-box_testing> [Accessed 6/3/2012].
4. 黑盒测试, 2012. Baidu. [online] Available at: <http://baike.baidu.com/view/51274.htm> [Accessed 6/3/2012].
5. 白盒测试, 2012. Baidu. [online] Available at: <http://baike.baidu.com/view/51297.htm> [Accessed 6/3/2012].
6. Software Testing Glossary, 2012. Ap Test. [online] Available at: <http://www.aptest.com/glossary.html> [Accessed 10/3/2012].
7. Acceptance Testing, 2012. Wikipedia. [online] Available at: <http://en.wikipedia.org/wiki/Software_testing#Acceptance_testing> [Accessed 10/3/2012].