**Chapter 5: Application Testing Phase**

The *Testing* phase is one the most important one, where the developed solution, in which a lot of time and efforts were invested in, get to be tested. The *Testing* phase usually runs smoothly when all other phases were completed adequately. The *Testing* process is used to ensure that the solution fits the Client’s needs. At organizational level, the product testing may be responsibility of different actors within the business, such as Project Sponsor, Project Manager, Testing Lead, Functional Analyst, Subject Matter Experts (SMEs), and the Technical Staff (Niesse, 2018).

When testing a software, developed with in Agile methods, it is important to follow few principles as follows:

* Testing is continuous;
* Continuous feedback;
* Tests performed by the whole team;
* Decrease time of feedback response;
* Simplified & clean code;
* Less documentation;
* Test driven.

There are a few advantages of Agile Testing, such as Saving time and money, Reduction of documentation, Flexible and highly adaptable to changes, provides a way for receiving regular feedback from the end-user, and Better determination of issues through daily meetings (reqtest, 2018).

There are several different types of Testing, and this chapter is focused on explaining a few of them, and determine an option chosen for the developed application.

**5.1 White-Box Testing**

*White-Box* Testing, also called *Clear-Box* Testing, *Open-Box* Testing, *Glass-Box* Testing, *Transparent-Box* Testing, *Code-Based* Testing, or *Structural* Testing, is a software testing method that tests the internal structure, the design and implementation of the solution to be implemented. Inputs are chosen by the tester to verify paths throughout the code and to determine the appropriate outputs (STFa, 2020).

In order for such testing to be completed, the tester must have a good knowledge and understanding of the internal structure, the code of the software. It tests the logic of the software and it is generally applicable to lower levels of software testing (Jain, 2020).

**Advantages**:

* It can start at an early stage, which means that there is no need to wait for the GUI to be available;
* This procedure is more thorough and has the possibility to cover most paths.

**Disadvantages**:

* High skilled personnel with thorough knowledge of programming and implementation is required, since this testing can be very complex;
* It can become a burden, if the changes are too frequent;
* It is an expensive method

(STFa, 2020)

**5.2 Black-Box Testing**

*Black-Box* Testing is also called *Behavioural* Testing, *Opaque-box*, *Closed-box*, *Specification-based*, or *eye-to-eye* Testing (Software Testing Help, 2020). With this method, the structure, the design, and the implementation of the software, being tested, are not known to the tester. It is usually functional but can also be non-functional. The errors found with this type of testing can be within a few categories, such as Incorrect of missing functions, Interface errors, Errors in data structures or external database access, Behaviour, or performance errors. The name of this testing technique is given because the software program acts like a *black-box* in the eyes of the tester. Figure 5.1a show the structure of the Black-Box Testing (STFb, 2020).

Unlike the *White-Box* testing, the *Black-Box* Testing analyses the functionality of an application, according to its specifications and tests the behaviour of the software. It is appropriate to the higher levels of testing of software and can be done by trial and errors (Jain, 2020).

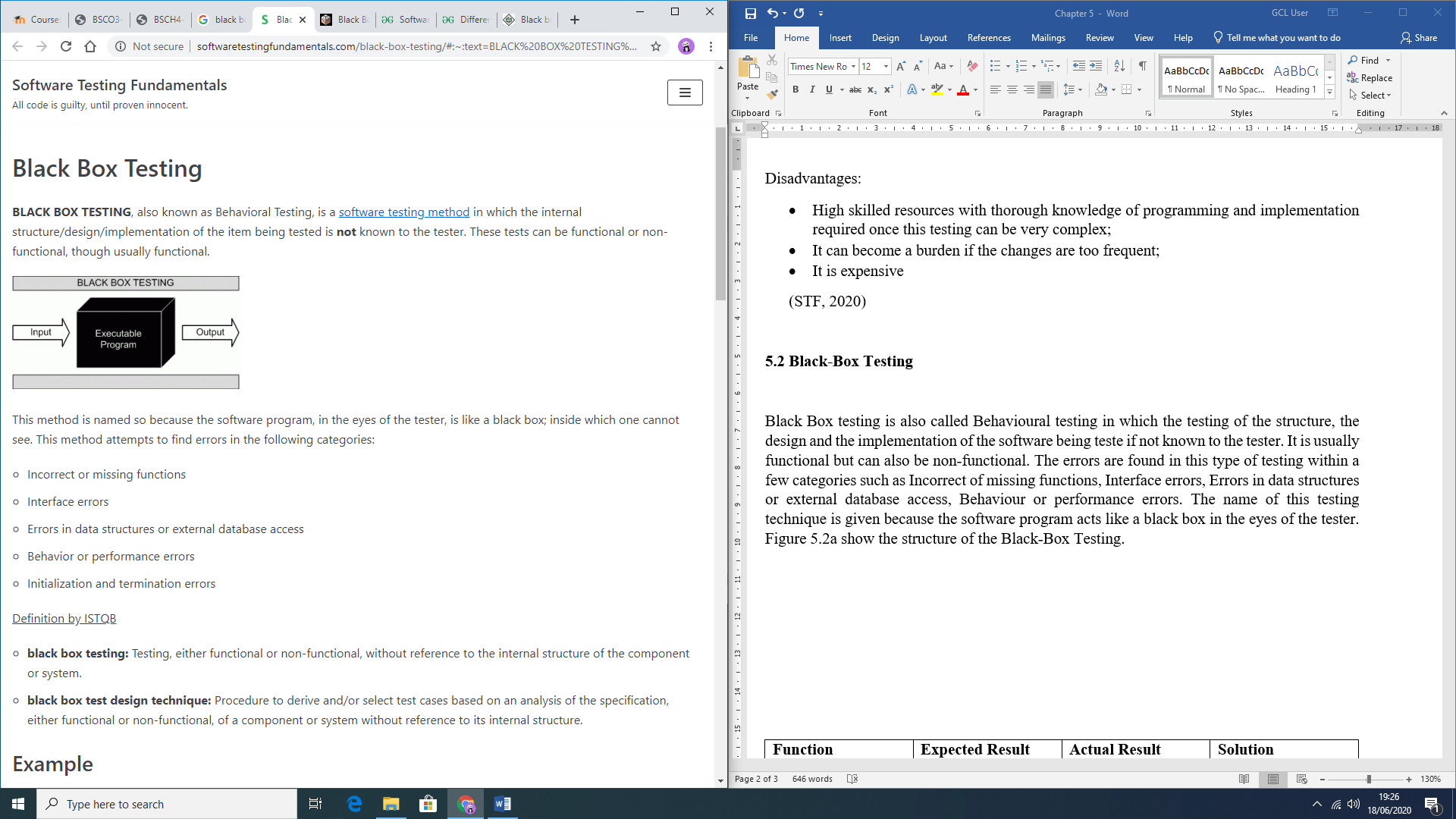
  
**Advantages**:

Figure 5.1a: Black-Box representation (STFb, 2020)

* The test is done from the user point of view, helping to expose discrepancies in the specifications;
* There is no need for knowledge of the programming languages or knowledge of the software;
* It can avoid developer-bias, since the testing can be conducted by a body, independent from the developers;
* Allows for an objective perspective;
* Once the specifications are completed, the test can be designed.

**Disadvantages**:

* Possibility of leaving many program paths untested, since it allows for just a small number of possible inputs to be tested;
* It can be difficult to design Testing Cases, if there are no clear specifications;
* Redundancy, if the developer has already run a Testing Case;

(STFb, 2020)

**5.3 Grey-Box Testing**

*Grey-Box* Testing combines both, *Black-Box* and *White-Box* Testing methods features. Initially, the internal structure of the software is known, which means involving having access to internal data structures and algorithms in order to design the Test Cases, but testing is done from the user’s point of view (Black-Box level). The name is given because the program is like a grey or semi-transparent box in the eyes of the tester, which means that internally can be partially seen (STFc, 2020).

By being able to identify context-specific errors that belong to web systems, if the testers encounter any defect, they are able to make the changes in the code and test it again in real time. It provides opportunity to test either, the presentation layer, and/or the internal coding structure of the software (JavaTPoint, 2020).

**Advantages**:

* The goals are clear for the users and developers while testing;
* User focused, since the testing is mostly done from user perspective;
* No need for high programming knowledge for this testing;
* Non-intrusive;
* The overall quality of the software is improved;
* Combined benefits from Black-Box and White-box testing;
* Unbiased, avoiding conflicts between a tester and a developer;
* More effective in integration testing.

**Disadvantages**:

* If testing is performed for distributed systems, the defect association is difficult;
* Limited access for code path traversal lead by limited access to internal structure;
* Not possible to do a complete White-box testing, because the source code cannot be accessed
* Not suitable for algorithm testing;
* Difficulty to design most of the Testing Cases.

(GeeksforGeeks, 2020)

**5.4 Chosen Testing Method**

*Grey-Box* testing was chosen for this project due to its combined advantages and to the fact that allows testing from both, user and developer perspectives. Errors are meant to be encountered while developing the software, however, with *Grey-Box* testing these errors can be fixed along with the coding. Also, the testing coverage is increased within all areas of the application and suits the limited project time frame.

**5.5 Errors Encountered During Application Development**

A few errors were encountered during the development of the software. Table 5.5a presents these errors and how they were fixed.

|  |  |  |  |
| --- | --- | --- | --- |
| **Function** | **Expected Result** | **Actual Result** | **Solution** |
| Register | Error message displayed, when user presses the *Register* button, *without* complying with security requirements | User registered successfully *even* without complying with security requirements | Use of the appropriate patterns (phone, e-mail, and password) and use of appropriate *validation* methods |
| Use of the filters | Necessary to use one filter, to activate the next, and so forward | User was able to select different filters *without* dependency | Nest each one of the filters within the other |
| Shelter Profile display | Show the appropriate information for specific shelter | *Mismatched* information displayed on the screen | Passing the specific information about a shelter through use of (putExtra()) in an *intent* |
| Return to *Animal* Profile from a *Shelter* profile | Returning to the *Animal* profile | Error occurred, landing the user to the *Home* screen | Passing the *Animal* object through an *intent* when opening the *Shelter* profile screen |

*Table 5.1a: Initial Errors Encountered during the Development process*

**5.6 Application Testing Process with Testing Cases**

All functionalities were completely tested and Table 5.1b presents the checks done with Testing Cases and outcomes of these tests.

Table 5.1b: Application Testing process with Testing Cases

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Action** | **Expected result** | **Pass** | **Fail** | **Actual result** |
| Open  application | Initial screen opens if user has *not registered* and *logged in*, otherwise main screen opens |  |  | Initial screen is displayed if user has *not registered* and *not logged in*. Main screen is displayed otherwise |
| Select *Register* option | *Registration* screen opens |  |  | *Registration* screen is displayed |
| *Register* button pressed *without* any data entered | Display *error message* to enter data for each field |  |  | Error message *is displayed* in each field, asking user to enter data |
| *Register* button pressed when *invalid* data is entered | Display *specific error* message to each field where data is *invalid*, guiding the user on how to fix it |  |  | *Specific error message* is displayed to each field where data is *invalid*, guiding the user on how to fix it |
| *Register* button pressed with *valid* data entered in all fields | Create *new user* and open *main* screen |  |  | New user is created, and main screen is displayed |
| Select *Login* option | *Login* screen opens |  |  | *Login* screen is displayed |
| *Login* button pressed *without* any data entered | Display *error message* to enter data |  |  | *Error message* is displayed in each field, asking user to *enter* data |
| *Login* button pressed when *invalid* data is entered | Display *specific Toast message,* regarding the error |  |  | Specific Toast message regarding the error is displayed |
| *Login* button pressed with *correct* username and password | User is logged in successfully, while a Toast messaged is displayed and *main* screen opens |  |  | Login is *successful* while a Toast message is displayed, and *main* screen is displayed |
| *“Not registered yet?”* TextView is clicked | *Registration* screen opens |  |  | *Registration* screen is displayed |
| *“Forgot your password?”* TextView is clicked | *Reset password* screen opens |  |  | *Reset password* screen is displayed |
| *Reset password* button is clicked *without* any e-mail entered | Display *error message* to enter username |  |  | *Error message* is displayed, asking user to *enter* their username |
| *Reset password* button is clicked with *invalid* e-mail entered | Display *specific error* message, guiding user on how to fix it |  |  | *Specific error* message is displayed, guiding user on how to fix it |
| *Reset password* button is clicked with *correct* e-mail entered | An e-mail with a *link for resetting the password* is sent to the user’s registered e-mail |  |  | Send a *resetting password link* to the user’s registered e-mail address |
| *Update Profile* menu item is clicked | *Update Profile* screen opens |  |  | *Update Profile* screen is displayed |
| *Update Profile* button clicked with *empty* fields | Display *error message* to enter data for each field |  |  | *Error message* is displayed in each field, asking user to enter data |
| *Update Profile* button clicked when *invalid* data is entered | Display *specific* error message to each field, where data is *invalid*, guiding the user on how to fix it |  |  | *Specific error message* is displayed to each field where data is *invalid*, guiding the user on how to fix it |
| *Update Profile* button clicked when *valid* data is entered | Display Toast message, notifying the user that the *changes were saved* |  |  | Toast message notifying the user that the *changes were saved* is displayed |
| *Logout* menu item is clicked | User is *logged out* while a Toast message is being displayed on the screen, and *initial* screen opens |  |  | *Logout successful* Toast message is displayed, and *initial* screen is displayed |
| *Category* filter is selected | Options of *Animal category* are displayed in a *dropdown menu* |  |  | Displays the *Animal category options* available in a *dropdown menu* format |
| *Category option* item is clicked | *List of* *animals* displayed on the screen to match the *selected category*. *Size filter* selection is enabled |  |  | Displays the *list of animals* that matches the *selected category* and enables the *size filter* selection |
| *Size filter* is clicked | Options of *size* are displayed in a *dropdown menu* |  |  | Displays the *size options* in a *dropdown menu* format |
| *Size option* item is clicked | *List of animals* displayed on the screen to match the *selected size*, while also matching the *pre-selected category* filter. *Breed filter* selection is enabled |  |  | Displays the *list of animals* that match the *selected size* while also matching the *pre-selected category* filter. Enables the *breed filter* selection |
| *Breed filter* is clicked | *Options of breeds* that match both the *pre-selected category* and *size filters* are displayed in a *dropdown menu* |  |  | Displays the *animal breed options* that match both the *pre-selected category* and *size filters* in a *dropdown menu* format |
| *Breed option* item is clicked | *List of animals* displayed on the screen to match *all filters applied* (category, size, and breed) |  |  | Displays the *list of animals* that match *all filters selected* (category, size, and breed) |
| *Chosen Animal* item from the list is clicked | *Animal profile* screen opens |  |  | *Animal profile* screen is displayed |
| Click on *Send Intent for Adoption* | Prompts the user to *select* from e-mail providers, available on the user’s phone, that may be used to send the Intent form |  |  | A selection of *e-mail providers*, available on the user’s phone is prompted, so the user can *choose* one to *send* the Adoption Intent form |
| E-mail provider is *selected* | Opens the external *e-mail application* with automatically generated template with *recipient’s address*, *subject* and *pre-defined e-mail body* with selected *animal name* information |  |  | External *e-mail application* is opened, with automatically generated template with *recipient’s address*, *subject* and *pre-defined e-mail body* with selected *animal name* information |
| *Shelter name* TextView is clicked | *Shelter profile* screen opens |  |  | *Shelter profile* screen is displayed |
| *Zoom in* button on the integrated *Google Map* is clicked | Map is *zoomed in* one level, being able to zoom in *up to level 20* |  |  | *Zoom in* one level is applied on map, being able to zoom in *up to level 20* |
| *Zoom out* button on the integrated Google Map is clicked | Map is *zoomed out* one level, being able to zoom out up to the *level 7* (starting on the minimum zoom level) |  |  | *Zoom out* one level is applied on map, being able to zoom out *up to the level 7* (starting on the minimum zoom level) |
| *Go back* to *Animal profile* button is clicked | Returns user back to the *Animal profile* screen |  |  | Goes *back* to the *Animal profile* screen, the user was navigated from |
| *Back* button is clicked | Return to the *previous* screen |  |  | Goes *back* to the *previous* screen |

**5.7 Chapter Summary**

Three major Testing methodologies were presented in this chapter (*White-Box* testing, *Black-Box* Testing and *Grey-Box* Testing) with their advantages and disadvantages, in order to decide on which one to be used within the current project. After analysing thoroughly each option, decided was that the *Grey-Box* Testing would be the most appropriate option, since it combines both the *Black-* and *White*-Box advantages and favours the short time frame of the Project.

Also presented in this chapter was the Testing process with Testing Cases, developed for the completed application, all of which were *successfully* passed.

The next chapter will present the end-user Testing results of the software (User Evaluation Results Analysis), as well as Project’s Conclusion, Achievements and Scope for Future work.

**References**

GeeksforGeeks, 2020. *GeeksforGeeks.* [Online]   
Available at: https://www.geeksforgeeks.org/gray-box-testing-software-testing/  
[Accessed 18 June 2020].

Jain, M., 2020. *GeeksforGeeks.* [Online]   
Available at: https://www.geeksforgeeks.org/differences-between-black-box-testing-vs-white-box-testing/  
[Accessed 18 June 2020].

JavaTPoint, 2020. *JavaTPoint.* [Online]   
Available at: https://www.javatpoint.com/grey-box-testing  
[Accessed 18 June 2020].

Niesse, E., 2018. *HRchitect.* [Online]   
Available at: https://hrchitect.com/testing-phase-clients-role-implementation/  
[Accessed 16 June 2020].

reqtest, 2018. *Reqtest.* [Online]   
Available at: https://reqtest.com/testing-blog/agile-testing-principles-methods-advantages/  
[Accessed 17 June 2020].

Software Testing Help, 2020. *Software Testing Help.* [Online]   
Available at: https://www.softwaretestinghelp.com/black-box-testing/  
[Accessed 18 June 2020].

STFa, 2020. *Software Testing Fundamentals.* [Online]   
Available at: http://softwaretestingfundamentals.com/white-box-testing/#:~:text=WHITE%20BOX%20TESTING%20(also%20known,is%20known%20to%20the%20tester.  
[Accessed 17 June 2020].

STFb, 2020. *Software Testing Fundamentals.* [Online]   
Available at: http://softwaretestingfundamentals.com/black-box-testing/#:~:text=BLACK%20BOX%20TESTING%2C%20also%20known,%2Dfunctional%2C%20though%20usually%20functional.  
[Accessed 18 June 2020].

STFc, 2020. *Software Testing Fundamentals.* [Online]   
Available at: http://softwaretestingfundamentals.com/gray-box-testing/  
[Accessed 18 June 2020].