

ASSIGNMENT

**PROJECT DOCUMENTATION**

by

Jai Prakash Advani

(1847229)

To

Fr. Justin P Varghese

Assignment submitted in partial fulfillment of the requirements for the award of degree of Master of Computer Applications of CHRIST (Deemed to be University

September– 2019

**DISEASE RECOGNITION SYSTEM FOR DOGS**

**TITLE:B FOR BARK**

**INTRODUCTION**

**PROPOSED SYSTEM**

**Disease recognition system for dogs** is web based application where one can search for diseases related to dog according to the symptoms provided by the user. The audience for this system is huge as in today’s world every age group has easy access to internet and every other person is concerned about their pets health.

With this system they can just log in and check whether the symptoms visible are something to be given importance or not. This system could save lives if its accuracy is maximised and if it has a huge databank of diseases and its symptoms.

**BENEFITS OF PROPOSED SYSTEM**

Our system automatically searches and give the outcome. In the Existing framework we have to manually search the details about the concerned disease which is not feasible for the user in most of the occasions.

* In our system we provide the user with the user friendly interface which gives substantially better outcome in detail.
* We also provide the details about the doctor, location of the concerned doctor and also the availability of the doctor.
* Furthermore, we are additionally exhibiting the trend analysis about the searched disease with a graphical portrayal.

**. Testing techniques**

1. **UNIT TESTING**

Unit testing is a software development process in which the smallest testable parts of an application, called units, are individually and independently scrutinized for proper operation. Unit testing is often automated but it can also be done manually. Unit testing involves only those characteristics that are vital to the performance of the under test. Once all of the units in a program have been found to be working in the most efficient and error-free manner possible, larger components of the program can be evaluated by means of integrating testing.

1. **INTEGRATION TESTING**

Integration testing is the phase in software testing in which individual software modules are combined and tested as a group. It occurs after unit testing and validation testing. Integration testing takes as its input modules that have been unit  tested, groups them in larger aggregates, applies tests defined in an integration test plan to those aggregates, and delivers as its output the integrated system ready for system testing.

**3.1 White Box Testing**

White-box testing, sometimes called glass-box testing, is a test-case design philosophy that uses the control structure described as part of component-level design to derive test cases. Using white-box testing methods, you can derive test cases that (1) guarantee that all independent paths within a module have been exercised at least once, (2) exercise all logical decisions on their true and false sides, (3) execute all loops at their boundaries and within their operational bounds, and (4) exercise internal data structures to ensure their validity.

**3.1.1 Types of White Box Testing**

**3.1.1.1 Basic path testing**

The basic path method enables the test-case designer to derive a logical complexity measure of a procedural design and use this measure as a guide for defining a basis set of execution paths. Test cases derived to exercise the basis set are guaranteed to execute every statement in the program at least one-time during testing.

**3.1.1.2 Flow graph notation**

The flow graph depicts logical control flow using the notation. Each structured construct has a corresponding flow graph symbol. And the testing is carried out like the traversal among the nodes.

**3.1.1.3 Independent program path**

An independent path is any path through the program that introduces at least one new set of processing statements or a new condition. When stated in terms of a flow graph, an independent path must move along at least one edge that has not been traversed before the path is defined.

**3.2 Black Box Testing**

Black-box testing, also called behavioural testing, focuses on the functional requirements of the software. That is, black-box testing techniques enable you to derive sets of input conditions that will fully exercise all functional requirements for a program. Black-box testing attempts to find errors in the following categories: (1) incorrect or missing functions, (2) interface errors, (3) errors in data structures or external database access, (4) behaviour or performance errors, and (5) initialization and termination errors. By applying black-box techniques, you derive a set of test cases that satisfy the following criteria: (1) test cases that reduce, by a count that is greater than one, the number of additional test cases that must be designed to achieve reasonable testing, and (2) test cases that tell you something about the presence or absence of classes of errors, rather than an error associated only with the specific test at hand.

**3.2.1 Types of Black Box Testing**

**3.2.1.1 Graph- based Testing Methods**

Graph based testing begins by creating a graph of important objects and their relationships and then devising a series of tests that will cover the graph so that each object and relationship is exercised and errors are uncovered. You can then derive test cases by traversing the graph and covering each of the relationships shown. These test cases are designed in an attempt to find errors in any of the relationships. Bezier describes a number of behavioural testing methods that can make use of graphs:

**1. Transaction flow modelling**. The nodes represent steps in some transaction, and the links represent the logical connection between steps. The data flow diagram can be used to assist in creating graphs of this type.

**2. Finite state modelling**. The nodes represent different user-observable states of the software, and the links represent the transitions that occur to move from state to state. The state diagram can be used to assist in creating graphs of this type.

**3. Data flow modelling**. The nodes are data objects, and the links are the transformations that occur to translate one data object into another.

**4. Timing modelling**. The nodes are program objects, and the links are the sequential connections between those objects. Link weights are used to specify the required execution times as the program executes.

**3.2.1.2 Equivalence Partitioning**

Equivalence partitioning is a black-box testing method that divides the input domain of a program into classes of data from which test cases can be derived. An ideal test case single-handedly uncovers a class of errors that might otherwise require many test cases to be executed before the general error is observed. Test-case design for equivalence partitioning is based on an evaluation of equivalence classes for an input condition. An equivalence class represents a set of valid or invalid states for input conditions. Typically, an input condition is either a specific numeric value, a range of values, a set of related values, or a Boolean condition.

**3.2.1.3 Boundary Value Analysis (BVA)**

A greater number of errors occurs at the boundaries of the input domain rather than in the center. It is for this reason that BVA has been developed as a testing technique. BVA leads to a selection of test cases that exercise bounding values. BVA is a test-case design technique that complements equivalence partitioning. Rather than selecting any element of an equivalence class, BVA leads to the selection of test cases at the edges of the class. Rather than focusing solely on input conditions, BVA derives test cases from the output domain as well.