2.1 GENERAL CONCEPTS

2.1.1 INTRODUCTION

This section defines the main techniques used in this project to fill full the objectives that the project must address. It also shows the feasibility study of applying model-based testing.

2.1.2 STATE MACHINE

A finite-state machine (FSM) is a mathematical model of computation used to design both computer programs and sequential logic circuits. It is conceived as an abstract machine that can be in one of a finite number of states.

The machine is in only one state at a time; the state it is in at any given time is called the current state. It can change from one state to another when initiated by a triggering event or condition; this is called a transition.

A particular FSM is defined by a list of its states, and the triggering condition for each transition, it allows to modeling the system under test before generating of test cases [3].

While sequence diagrams are used to describe the interaction between objects, state machines are used to define the behavior of one object (or a class of objects) [4].

A state machine can be described as:

- An initial state or record of something stored someplace
- A set of possible input events
- A set of new states that may result from the input
- A set of possible actions or output events that result from a new state

A finite state machine can be used both as a development tool for approaching and solving problems and as a formal way of describing the solution for later developers and system maintainers. There are a number of ways to show state machines, from simple tables through graphically animated illustrations [5].

2.1.3 BLACK-BOX TESTING

Black box testing is a software testing techniques in which functionality of the software under test (SUT) is tested without looking at the internal code structure implementation details and knowledge of internal paths of the software. This type of testing is based entirely on the software requirements and specifications.

Black box testing method is applicable to all levels of the software testing process:

- Unit Testing
- Integration Testing
- System Testing
- Acceptance Testing

This method of attempts to find errors in the following categories:

- Incorrect or missing functions
- Interface errors
- Errors in data structures or external database access
- Behavior or performance errors
- Initialization and termination errors[6]

2.1.3.1 MODEL-BASED TESTING (MBT)

model-based testing is the generation of executable test cases that include oracle information, such as the expected output values of the SUT, or some automated check on the actual output values to see if they are correct. This is obviously a more challenging task than just generating test input data or test sequences that call the SUT but do not check the results. To generate tests with oracles, the test generator must know enough about the expected behavior of the SUT to be able to predict or check the SUT output values. In other words, with this definition of model based testing, the model must describe the expected behavior of the SUT, such as the relationship between its inputs and outputs. But the advantage of this approach is that it is the only one of the four that addresses the whole test design problem from choosing input values and generating sequences of operation calls to generating executable test cases that include verdict information [6].

2.1.3.2 MODEL JUNIT

JUnit is a unit testing framework for the Java programming language. Units are the smallest module of functionality in a computer program. These are usually in the form of a method. Therefore, JUnit is most commonly used to

test the functionality of individual methods. Experience with JUnit has been important in the development of Test-Driven Development [8].

ModelJUnit is a Java library that extends JUnit to support model-based testing. Models are extended finite state machines that are written in a familiar and expressive language: Java. ModelJUnit is an open source tool.

ModelJUnit allows you to write simple finite state machine (FSM) models or extended finite state machine (EFSM) models as Java classes, then generate tests from those models and measure various model coverage metrics [9].

2.1.4 ECLIPSE

Eclipse is a multi-language software development environment comprising a base workspace and an extensible plug-in system for customizing the environment. It is written mostly in Java. It can be used to develop applications in Java and, by means of various plug-ins, other programming languages including Ada, C, C++, COBOL, Fortran, Haskell, Perl, PHP and Python. It can also be used to develop packages for the software Mathematical. Development environments include the Eclipse Java development tools (JDT) for Java and Scala, Eclipse CDT for C/C++ and Eclipse PDT for PHP, among others.

The initial codebase originated from IBM Visual Age. The Eclipse software development kit (SDK), which includes the Java development tools, is meant for Java developers. Users can extend its abilities by installing plug-ins written for the Eclipse Platform, such as development toolkits for other programming languages, and can write and contribute their own plug-in modules [10].

2.1.5 UML LANGUAGE

UML stands for “Unified Modeling Language”. It is an industry-standard graphical language for analysing, describing and documenting the artifacts of an object-oriented system under development or (under test). The UML uses mostly graphical notations to express the OO analysis and design of software projects.

The SUT analyzed using Use case, Sequence, Activity, Class and Statechart diagrams to emphasize what must happen in the application being modeled. These behavior and structural diagrams illustrate the behavior and structure of a SUT [11].

2.1.5.1 ENTERPRISE ARCHITECT

Is UML modeling tool to modify the diagrams that help in analysis process.

2.1.5.1.1 USE CASE DIAGRAM

A use case is a description of some way in which a system or a business is used, by its customers, users or by other systems. Mainly used for capturing user requirements (describing a set of user scenarios) and it's Work like a contract between end user and software developers.

2.1.5.1.2 CLASS DIAGRAM

Class diagram in UML is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects.

2.1.5.1.3 SEQUENCE DIAGRAM

Sequence diagrams are used to represent the flow of messages, events and actions between the objects or components of a system.

The horizontal dimension shows the objects participating in the interaction, and the vertical arrangement of messages indicates their order.

2.1.5.1.4 ACTIVITY DIAGRAM

Activity Diagrams are a type of flowchart used to describe business process or workflow of a system; that is, they show the flow of control from activity to activity in the system, what activities can be done in parallel, and any alternate paths through the flow**.**

2.1.5.1.5 STATECHART DIAGRAM

The statechart diagram models the different states that a class can be in and how that class transitions from state to state. It can be argued that every class has a state, but that every class shouldn't have a statechart diagram. Classes with three or more potential states during system activity should be modeled.

2.2 PREVIOUS STUDIES

The concept of Model Based testing is a brand new approach in software testing and quality assurance techniques. Because of which, there are no major (real) software products adopted the concept of MBT yet.

Manual testing is the current software testing technique in use in most of software production companies.

2.3 FEASIBILITY STUDY

The feasibility study is an evaluation and analysis of the potential of a proposed project which is based on extensive investigation and research to support the process of decision making.

A feasibility study main goal is to assess the economic viability of the proposed business. The feasibility study needs to answer the question: “Does the idea make economic sense?”[12].

2.3.1 ECONOMIC FEASIBILITY STUDY

Analysis of a project costs and revenues in an effort to determine whether or not it is possible to complete.

2.3.1.1 NET PRESENT VALUE ANALYSIS

Net present value determines the profitability of the project in terms of today's dollar values. Will tell you that if you invest in the proposed project, after n years you will have $XXX profit/loss on your investment.

2.3.1.1.1 NET PRESENT VALUE WITH USING MANUAL TESTING

	Year 0	Year 1	Year 2	Year 3	Year 4	total
Cost						
Development cost	100,000					
Operation and maintenance cost		15,000	20,000	25,000	30,000	
Discount factor for 12%	1.000	0.893	0.797	0.712	0.636	
PV for annual cost	100,000	13,395	15,940	17,800	19,080	
Total PV of lifetime costs						**166,215**
Benefits						
Derived from operation	0	50,000	65,000	70,000	80,000	
Discount factor for 12%	1.000	0.893	0.797	0.712	0.636	
PV for annual benefits	0	44,650	51,805	49,840	50,880	
Total PV of lifetime benefits						**197,175**
Net present value						**30,960**

The net present value of the investment in the project after 5 years is 30,960.

2.3.1.1.2 NET PRESENT VALUE WITH USING MODEL-BASED TESTING

	Year 0	Year 1	Year 2	Year 3	Year 4	total
Cost						
Development cost	100,000					
Operation and maintenance cost		10,000	15,000	20,000	25,000	
Discount factor for 12%	1.000	0.893	0.797	0.712	0.636	
PV for annual cost	100,000	8,930	11,955	14,240	15,900	
Total PV of lifetime						**151,025**

costs						
Benefits						
Derived from operation	0	50,000	65,000	70,000	80,000	
Discount factor for 12%	1.000	0.893	0.797	0.712	0.636	
PV for annual benefits	0	44,650	51,805	49,840	50,880	
Total PV of lifetime benefits						**197,175**
Net present value						**46,150**

The net present value of the investment in the project after 5 years is 46,150.

The net present value of the project when using model-based testing is greater than using manual testing, the reason is that model-based testing aims to test the model of the system before coding process and detect bugs and errors early; that leads to reduction of maintainability after implementation of the system and hence reduction of total costs.

2.4 CONCLUSION

This chapter explained the technologies and techniques that used in this project; it also showed that there are not previous studies for this field yet and proved that the model based testing saves costs and that is illustrated in the feasibility study of using model-based testing result**.**

The next chapter will describe the structure and analysis of the salary system case study using UML diagrams by enterprise architect tools**.**