DESIGN OF A MODEL –DRIVEN DECISION SUPPORT PLATFORM FOR QUEUE MANAGEMENT

BY

HARUNA, TOFA
(M.TECH/OR/09/0143)

A THESIS SUBMITTED TO THE DEPARTMENT

OF

STATISTICS AND OPERATIONS RESEARCH,

SCHOOL OF PURE AND APPLIED SCIENCES,

MODIBBO ADAMA UNIVERSITY OF TECHNOLOGY,

YOLA

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE AWARD OF THE DEGREE OF MASTER OF TECHNOLOGY (M.TECH) IN OPERATIONS RESEARCH

SEPTEMBER, 2012.

DECLARATION

I hereby declare that this thesis titled “Design of a Decision Support Platform for Queue Management” was written by me and it is a record of my own research work. It has not been presented before in any previous application for a higher degree. All references cited have been duly acknowledged.

..……..... …….....

Haruna , Tofa **Date**

DEDICATION

This project work is first and foremost dedicated to God Almighty, for Life, protection, wisdom and understanding thus far, mostly for giving me a second chance to live

This work is especially dedicated to my late brother Labune Haruna, whose short and impactful life I will forever cherish till we meet to part no more.

ACKNOWLEDGEMENT

I acknowledge the encouragement and support of Hon. (Dr) Hassan Ahmadu Haruna, Hon. Nuhu Poloma, and Dr. D.K. Diraso, all being there for me at the very beginning of it all.

I am grateful to my Supervisor, Dr. I. J. Dike for his down-to – earth supervision, concerted perusal, sacrifice of his convenience, and for bearing with my short comings during the course of this project work. I'm grateful.

To my friends, Kefas Peter Garba, Kehinde Egunsola, Ikili "Oga" Bello and the most beloved Mrs Yimmi T. Haruna.

I acknowledge the contributions of Dr. S.S. Abdulkadir and Dr. A.K. Mishra for the kind guidance, encouragement and unequalled support on issues of life and academics.

Finally, to my dear son, God's special gift, source of joy and peace, little Salem T. Haruna.

ABSTRACT

Queues are common features not only in our daily-life situations such as at banks, postal offices, in public transportation or hospitals but also in more technical environments, such as in manufacturing, computer networking and telecommunications. In general, a queue is formed in a system when either number of customers exceeds the number of service facilities, or service facilities do not work efficiently. The problem of queues has been solved in many different ways using various mechanical, electronic and computerized systems, which are usually customized for specific systems and do not support in-depth queue analysis, data management, nor support resource allocation decisions to meet customer demand.

In this project report, we discuss the design and implementation of a model- driven decision support platform for queue management that facilitates the specification, management, and analysis of queuing systems across diverse queuing systems and types. The decision support system software developed constitutes a solution to provide instantaneous accurate data that can be used to support resource allocation decisions in queuing situations. The system which implements models from queuing theory is composed of five subsystems which make up the decision support system that can be installed on a standalone system or a Client Server architecture enabling partial or total access to the database and providing real time data for decision making.

The application designed using the C# language is hosted on the Microsoft .NET framework 4.5 platform running on the Windows OS. The database is relational designed and supported on the Microsoft SQL server. The framework used in the design and modelling of the system is the Object-Oriented methodology which allows the use of principles such as inheritance, polymorphism and modularity. The system has been successfully tested and implemented on queuing systems in Healthcare, Banking and Retail marketing among others.

The results presented in this paper indicate that huge benefits are possible from the development of decision support systems based on existing mathematical models to simplify the task of decision making while simultaneously giving them greater usage and applicability.

TABLE OF CONTENTS

		Pages
Title Page		i
Declaration		ii
Certification		iii
Dedication		iv
Acknowledgement		v
Abstract		vi
Table of Contents		vii
CHAPTER ONE		
INTRODUCTION		
1.0.0	Background of the Study	1
1.2.0	Statement of problems	5
1.3.0	Aims and Objectives of the Study	6
1.4.0	Significance of the Study	6
1.5.0	Scope and Limitations of the Study	7
CHAPTER TWO		
REVIEW OF RELATED LITERATURES		
2.0.0	Preamble	8
2.1.0	Queuing Theory overview	8
2.1.1	Notations of Queuing Theory	9
2.1.2	Model Construction and Analysis	9
2.1.3	Single Server Queues	10
2.1.4	Poisson Arrival & Service	10
2.1.5	Poisson Arrival & General Service	11
2.1.6	Multi Server Queues	11
2.1.7	Infinitely Many Servers	11
2.2.0	The Birth and Death Process	12
2.2.1	The Birth & Death Process in Queuing Theory	12
2.2.2	M/M/1 Queuing Model	12
2.2.3	M/M/C Queuing Model	13
2.2.4	M/M/1/K Queuing Model	13
2.2.5	Equilibrium	14
2.2.6	The Limit Behaviour	14
2.3.0	Decision Support System Overview	14
2.3.1	Definition of Decision Support Systems	15
2.3.2	History of Decision Support System	16
2.4.0	Decision Support System Theory Development	19
2.5.0	Decision Support System Application Development	21
2.5.1	Model-Driven Decision Support System	21
2.5.2	Data-Driven Decision Support System	22
2.5.3	Communication -Driven Decision Support System	24
2.5.4	Knowledge-Driven Decision Support System	26
2.5.5	Web-Driven Decision Support System	27

2.6.0 Decision Support System Development Framework 28
2.7.0 Classification of Decision Support System 29
2.8.0 Applications of Decision Support System 30
2.9.0 Benefits of Decision Support System 30
2.10.0 Characteristics of Decision Support System 31

CHAPTER THREE
RESEARCH METHODOLOGY

3.0.0 Introduction 33
3.1.0 System Analysis 33
3.2.0 Design Objectives 34
3.3.0 Research Approach 35
3.4.0 Research Methodology 36
3.5.0 Program Design 37
3.5.1 User Login/ Authentication Module 37
3.5.2 Application Setup Module 37
3.5.3 User Account Setup Module 37
3.5.4 Industry/ Application Setup Module 38
3.6.0 Queue Analysis Module 38
3.7.0 File Design and Specification 38
3.7.1 Authentication Module Database File Design 38
3.7.2 User Account Setup Module Database File Design 39
3.7.3 Industry Module Database File Design 39
3.7.4 File Organization 39
3.8.0 Decision Support System Design Tasks 40
3.8.1 Application Architecture Design 40
3.8.2 System Database Design 40
3.8.3 System Interface Design 41
3.8.4 User Interface Design 41
3.9.0 System Design Specification 41
3.9.1 Programming Tools 42
3.10.0 System Requirements Specifications 42
3.10.1 Hardware Specification 43
3.10.2 Software Requirements 43

CHAPTER FOUR
DATA PRESENTATIONS AND ANALYSIS/SYSTEM DEVELOPMENT
4.0.0 Introduction 44
4.1.0 System Development 44
4.1.1 Program Coding 44
4.1.2 Program Interface Design 44
4.1.3 Output Design 44
4.2.0 System Testing 45
4.3.0 System Installation 45
4.4.0 System Documentation 45

CHAPTER FIVE
SUMMARY, CONCLUSION, RECOMMENDATIONS AND SUGGESTIONS FOR FURTHER STUDIES

5.0.0	Summary of the study	46
5.1.0	Conclusion of the study	46
5.2.0	Recommendations	47
5.3.0	Suggestions for further study	47

References
Appendix A: Performance Measures of Queuing System
Appendix B: Program Code Listing
Appendix C: Interface design samples

CHAPTER ONE

INTRODUCTION

1.0.0 BACKGROUND OF THE STUDY

People spend time waiting in line at post offices, at shops, at clinics, at banks, and at many other places that provide services. Queues or waiting lines "are also found in workshops where the machines wait to be repaired; at a tool crib where the mechanics wait to receive tools; in a warehouse where items wait to be used, incoming calls wait to mature in the telephone exchange, trucks wait to be loaded, airplanes wait either to take off or land and so on. In general, a queue is formed at a queuing system when either customers (human beings or physical entities) requiring service wait due to the fact that the number of customers exceeds the number of service facilities, or service facilities do not work efficiently and take more time than prescribed to serve a customer." (Sharma, 2008)

The word 'queue' refers to customers waiting for service in a line. It may be a physical line of customers or a waiting list of items to be processed etc.

Much of this time spent in waiting lines is wasted because people are unable to do other useful things while queuing. Often it is a tiring and disagreeable activity - standing in line doing nothing!

Unfortunately, waiting lines cannot be done away with because unpleasant though it is, queuing in its most fundamental form brings forth the advantage of fairness to customers based on the order of their arrival. In other circumstances, queuing allows for the provision of better service or achievement of higher efficiency.

The problem of serving customers in a specific sequence at business or public establishments has been solved in many different ways. Depending on available resources and technology, various mechanical, electronic and computerized systems have been designed and implemented. Some examples of such systems are:

- **Physical Barrier Systems**: These are the earliest queue management methods used. The system aimed at guiding queue formation and organizing it in the most efficient way.
- **Signage and Signaling Systems:** These systems aim to provide information to people queuing to aid efficient queue formation and flow, as well as setting service expectations.

- **Automatic Queue Measurement Systems:** These systems use a variety of measurement technologies which predict and measure queue lengths and waiting times and provide management information to help service levels and resource deployment. (http://en.wikipedia.org/wiki/Queue...)

Sometimes the operations manager is faced with the decision of determining just how many service facilities should be in operation in order to adequately meet customer demand. Examples of this type of situation include determining the appropriate number of checkout clerks in a supermarket, tellers in a bank, doctors on duty in an emergency room, gas pumps in a service station and so on. Queuing theory is a quantitative technique useful in determining the optimum number of service facilities. (Sharma, 2008)

The first objective of any queue management system is to achieve a better quality of service to customers. This must be done taking into cognizance the fact that there are two basic types of costs associated with waiting line problems: First, there are the fairly tangible costs involved in operating each service facility like costs of equipment, materials, labor, etc. These costs, of course, rise as the number of service facilities put into operation increases. On the other hand, there are the relatively intangible costs associated with causing customers to have to wait in line for some period prior to being served- physical discomfort, adverse emotional reactions, reduced or lost sales and so on. Of course, as the number of service facilities in operation increases, the time the customer has to wait in line, on the average decreases, and hence so too do these costs.

Modern queue management systems trends are moving towards attempts to make use of computerized systems to produce statistical reports on information such as arrival rates and patterns, waiting and service times, and default and reneging cases. Some real time examples for this case can be customers waiting in the queue in banks or to buy groceries in departmental stores. The contribution of the computer here is to maintain the queue according to the arrival time of the event, in this case the customers, and process each event one after the other according to their arrival time.

Automatic queue management systems with numbered tickets appeared in the mid-20th century, and were used by a few public or social services and local authorities (the reception areas in the French social security were among the first).

The Call Forward system, with a joint waiting line for several reception points, also had its moment of glory in the 1990s in public administrations (e.g. the Post Office and SNCF in France) and is still used in the US and UK, mainly at supermarkets check-out .

Since they first appeared in the 1990s, more sophisticated computerized systems enabled to avoid static waiting lines have continued to develop, with an ever-increasing choice of often undreamed-of functionalities… (http://81.255.198.178/pdf/EN/ESII-WhitePaper_EN.pdf)

Another example is the M2K Digital Signage Queue Management System which is a system that is widely used in retail, hospital, and banks with traditional LED displaying the customers' waiting numbers. "M2KSYS introduces new implementation of Queue Management System that replaces the traditional LED counters with the powerful digital signage in-house marketing solution. It allows users to display their queue information on a LCD/Plasma display panel together with other information such as in house advertisement, Live TV and scrolling RSS or text messages."(http://www.m2ksys.com/documents/45.html)

In the case of a common queue guide for several call forward windows (cash, desks), a LED or video screen at the head of the queue displays the number called with a directional arrow. The number called is then shown by a number plate, arrow, floor-plan etc.

When waiting times may be long, a possible first stage is to use a beeper or text message or Smart phone message to send customers to a final waiting area, where they are free to move around the store or area. If the reception procedure takes several stages, or if additional waiting time is necessary before the provision of the service can be concluded, it should be possible to place visitors on hold for subsequent treatment.

Yet another implementation of the queue management systems is the Wavetec Queue Management System, eQ. The system tunes up the waiting area so that clients have a pleasant experience throughout by offering them a hassle-free and personalized service.

The eQ system is practical for facilities that require managing physical queues dynamically like embassies, airports, immigration centers or those with 'dispersed queues' to allow customers the freedom to browse the internet while waiting for their turn, for example, in super markets; and yet others allow their customers to catch a breath of air before serving them like banks, hospitals etc

This systematic approach towards crowd management gives more control to the front-end personnel, thereby empowering them to ensure dedicated service. eQ's state-of-the-art reporting Engine equips management with valuable information to measure their staff

performance against service standards for continuous improvement. eQ's audio component for announcement, flexibility in choice of language and content management allows for in-store promotions and better customer guidance, delighted customers, empowered employees and informed management." (http://www.wavetec.com/queue-management-system.html)

A more recent development in queue management systems is Q-Flow®, a real-time, web-enabled customer flow and queuing system tool for directing, organizing and optimizing the throughput of customers in any-sized organization. Consisting of both software and hardware elements, Q-Flow® handles appointments, reception, registration, customer routing and queuing, service documentation, monitoring, analysis and reports (www.acftechnologies.com).

It may be worth noting that most of these systems are digital electronics based hardware systems that make use of the computer to perform calculations, present report in digital forms without capabilities for in depth analysis like the 'WHAT IF" analysis.

The use of decision support systems in queue management is still in its infant stages and considerable research effort are being directed towards the possibilities of their implementation, as noted by Wilson et al, "the FAA is sponsoring research in NextGen Surface-Trajectory-Based Operations (STBO). Collaborative Departure Queue Management (CDQM), a decision support tool (DST) used to support surface tactical flow, is currently being demonstrated at two airports, neither of which experience frequent snow events... research in these areas will continue." (Wilson, Hurley and Diffenderfer, 2011)

In general, "Decision Support Systems combine models, data, and information technology tools for effective decision-making and are becoming increasingly popular in business and industry (Abhijit and Ravindra, 2007).

Potentially, innovative decision support systems (DSSs) can yield competitive advantage for an organization or at least maintain an organization's competitive position. Evidence indicates managers can now use sophisticated data-driven and document-driven DSSs to obtain information that was buried for many years in filing cabinets or archived on computer storage systems, and also in real time to support decision making. Model-driven DSSs can reduce waste in production operations and improve inventory management. Knowledge-driven DSSs can help managers evaluate employees or help technical staff diagnose problems.

The widespread use of computer technology has changed the way companies do business. Information technology has altered relationships between companies and their suppliers, customers and rivals.

Also, DSSs can create a major cost advantage by increasing efficiency or eliminating value chain activities. For example, a bank or mortgage loan firm may reduce costs by using a new DSS to consolidate the number of steps and minimize the number of staff hours needed to approve loans or to attend to customers. Technology breakthroughs can sometimes continue to lower process costs, and rivals who imitate an innovative DSS may nullify or remove any advantage.

Finally, decision support systems can be used to help a company better focus on a specific customer segment and hence gain an advantage in meeting that segment's needs. Management information systems and decision support systems can help track customers, and DSSs can make it easier to serve a specialized customer group with special services. Some customers won't pay a premium for targeted service, and larger competitors also target specialized niches using their own DSSs.

1.2.0 **PROBLEM STATEMENT**

Waiting lines or queues is a common occurrence both in everyday life and in variety of business and industrial situations and, waiting for service is typically a negative consumer experience and causes unhappiness, frustration, and anxiety.

Operation managers are faced with the problem of determining the optimal number of service facilities to be in operation in order to adequately meet customer demand especially in queuing situations.

Long waiting times due to presence of queues is usually perceived by customers as an evidence of inefficiency on the part of the organization which causes customer dissatisfaction and loss of customer goodwill.

Most of the present implementations of queue management systems are mostly digital electronics based LED hardware systems with little or no capabilities for data management and in-depth analysis of queuing situations to help organizations in making optimal policy and resource allocation decisions.

Many complex and potentially useful operational research and mathematical models for queue management have been developed but have remained largely unimplemented by industry

because of their mathematical complexity and 'user unfriendliness' to managers and industry stakeholders.

Many hardware based queue management system are usually customized for use in specific industries and do not possess the capabilities to provide support for optimal decision making under dynamic situations of today's competitive business world. For example a hardware queue management system designed for banks cannot be used for managing the queues that may arise in Telephone exchange systems.

Most of the hardware based queue management systems are not based proven and well analyzed queuing theory models as their operations are based on physical principles of digital sensors and counting machines.

1.3 AIM AND OBJECTIVES OF THE STUDY

The aim of this project work is to design a model-driven decision support platform for queue management with enablement for in-depth analysis of queuing situations, data management and which can be used for management queuing systems across various sectors.

The objectives of this work include:

1. To design and develop a decision support system to assist managers in making resource allocation decisions based on in-depth analysis of queuing systems
2. To develop a relatively affordable system that will improve the potential efficiency of firms by providing capabilities for effective queue management.
3. To provide a system that will serve as a bridge between complex operations research models usually found in the academics and the practical needs of organizations and industry.
4. To design and develop a potentially powerful queue management system which can be used to manage various queue situations across different sectors of industry.

1.4 SIGNIFICANCE OF THE STUDY

A study on the design of model-based decision support system for queue management will have a significant impact on the efficiency of many organizations facing queuing situations especially in the Banking, Healthcare and Aviation sectors of the Nigerian economy.

The proposed study has the potential to greatly reduce operating costs in many organizations and improve the efficiency of resource allocation problems by aiding decisions on the number of service facilities that are required for specific operation periods.

The proposed Model-Based Decision Support System with its implementations of proven queuing models, powerful data management capabilities and user friendly Graphical User Interface, will provide a desired platform for the translation of powerful mathematical models into user friendly applications to support management decision making.

The proposed study is designed to make academic contributions to knowledge on the concept of decision support systems with particular emphasis on application to queue management. It is therefore motivated by the desire to make scholarly contribution to the search for more workable approach to the management of queuing operations.

The proposal for the design and implementation of the model based decision support system for queue management is based on the belief that the introduction of the concept into queue management operations would have a revolutionary effect on the role operations managers, queuing systems (banks, factories, hospitals, supermarkets, airports, etc) and the nature of their activities.

The implementation of the proposed decision support system is anticipated to swap inefficiency for efficiency thereby strengthening the existing role of operations managers.

1.5 SCOPE OF THE STUDY

The proposed research will focus on the design and development of decision support software for queue management. The study will be restricted to the detailed design; programming and coding of the proposed model based decision support system.

CHAPTER TWO

LITERATURE REVIEW

2.0 PREAMBLE

This chapter reviewed related literature and theoretical framework for the proposed study. Therefore, concepts and terms that shed more light on this study are forthwith relayed, for a better understanding of the proposed study.

2.1.0 AN OVERVIEW OF QUEUING THEORY

Delays and queuing problems are most common features not only in our daily-life situations such as at a bank or postal office, at a ticketing office, in public transportation or in a traffic jam but also in more technical environments, such as in manufacturing, computer networking and telecommunications. They play an essential role for business process re-engineering purposes in administrative tasks. Queuing models provide the analyst with a powerful tool for designing and evaluating the performance of queuing systems.

Whenever customers arrive at a service facility, some of them have to wait before they receive the desired service. It means that the customer has to wait for his/her turn, may be in a line. Customers arrive at a service facility with several queues, each with one. The customers choose a queue of a server according to some mechanism e.g., shortest queue or shortest workload (Blanchard and Fabrycky, 1990)

Sometimes, insufficiencies in services also occur due to an undue wait in service may be because of new employee. Delays in service jobs beyond their due time may result in losing future business opportunities.

Queuing theory is the study of waiting in all these various situations. It uses queuing models to represent the various types of queuing systems that arise in practice. The models enable finding an appropriate balance between the cost of service and the amount of waiting.

In queuing theory, a queuing model is used to approximate a real queuing situation or system, so the queuing behavior can be analyzed mathematically. Queuing models allow a number of useful steady state performance measures to be determined, including:

- the average number in the queue, or the system,
- the average time spent in the queue, or the system,
- the statistical distribution of those numbers or times,
- the probability the queue is full, or empty, and

- The probability of finding the system in a particular state.

These performance measures are important as issues or problems caused by queuing situations are often related to customer dissatisfaction with service or may be the root cause of economic losses in a business. Analysis of the relevant queuing models allows the cause of queuing issues to be identified and the impact of proposed changes to be assessed.

2.1.1 NOTATION

Queuing models can be represented using Kendall's notation:

A/B/S/K/N/D

where:

- A is the inter-arrival time distribution
- B is the service time distribution
- S is the number of servers
- K is the system capacity
- N is the calling population
- D is the service discipline assumed

Many atimes the last members are omitted, so the notation becomes A/B/S and it is assumed that K = ∞, N = $\infty$and D = FIFO.

Some standard notations for distribution (A or B) are:

- M: for a Markovian (poisons, exponential) distribution
- E_κ : for an Erlang distribution with κ phases
- D: for degenerate (or deterministic) distribution (constant)
- G: for general distribution (arbitrary)
- PH: for a phase-type distribution

2.1.2 MODELS CONSTRUCTION AND ANALYSIS

Queuing models are generally constructed to represent the steady state of a queuing system, that is, the typical, long run or average state of the system. As a consequence, these are stochastic models that represent the probability that a queuing system will be found in a particular configuration or state.

A general procedure for constructing and analyzing such queuing models is:

1. Identify the parameters of the system, such as the arrival rate, service time, queue capacity, and perhaps draw a diagram of the system.
2. Identify the system states. (A state will generally represent the integer number of customers, people, jobs, calls, messages, etc. in the system and may or may not be limited.)
3. Draw a state transition diagram that represents the possible system states and identify the rates to enter and leave each state. This diagram is a representation of a Markov chain.
4. Because the state transition diagram represents the steady state situation between states, there is a balanced flow between states so the probabilities of being in adjacent states can be related mathematically in terms of the arrival and service rates and state probabilities.
5. Express all the state probabilities in terms of the empty state probability, using the inter-state transition relationships.
6. Determine the empty state probability by using the fact that all state probabilities always sum to 1.

Whereas specific problems that have small finite state models can often be analyzed numerically, analysis of more general models, using calculus, yields useful formulae that can be applied to whole classes of problems.

2.1.3 SINGLE-SERVER QUEUE

Single-server queues are, perhaps, the most commonly encountered queuing situation in real life. One encounters a queue with a single server in many situations, including business (e.g. sales clerk), manufacturing industry (e.g. a production line), transport (e.g. queues that the customer can select from) Consequently, being able to model and analyze a single server queue's behavior is a particularly useful thing to do.

2.1.4 POISSON ARRIVALS AND SERVICE

M/M/1/∞/∞ (Single Server Infinite Queue Model) represents a single server that has unlimited queue capacity and infinite calling population, both arrivals and service are Poisson (or random) processes, meaning the statistical distribution of both the inter-arrival times and the service times follow the exponential distribution. Because of the mathematical nature of the exponential distribution, a number of quite simple relationships can be derived for several performance measures based on knowing the arrival rate and service rate.

This is fortunate because an M/M/1(Single Server Infinite Queue Model) queuing model can be used to approximate many queuing situations.

2.1.5 POISSON ARRIVALS AND GENERAL SERVICE

M/G/1/∞/∞ (Single Server Infinite Queue with Unspecified Service Time Distribution) represents a single server that has unlimited queue capacity and infinite calling population, while the arrival is still Poisson process, meaning the statistical distribution of the inter-arrival times still follow the exponential distribution, the distribution of the service time does not. The distribution of the service time may follow any general statistical distribution, not just exponential. Relationships can still be derived for a (limited) number of performance measures if one knows the arrival rate and the mean and variance of the service rate. However the derivations are generally more complex and difficult.

A number of special cases of M/G/1 provide specific solutions that give broad insights into the best model to choose for specific queuing situations because they permit the comparison of those solutions to the performance of an M/M/1 model.

2.1.6 MULTIPLE-SERVERS QUEUE

Multiple (identical) - server queue situations are frequently encountered in telecommunications or a customer service environment. When modeling these situations care is needed to ensure that it is a multiple servers queue, not a network of single server queues, because results may differ depending on how the queuing model behaves.

One observational insight provided by comparing queuing models is that a single queue with multiple servers performs better than each server having their own queue and that a single large pool of servers performs better than two or more smaller pools, even though there are the same total number of servers in the system.

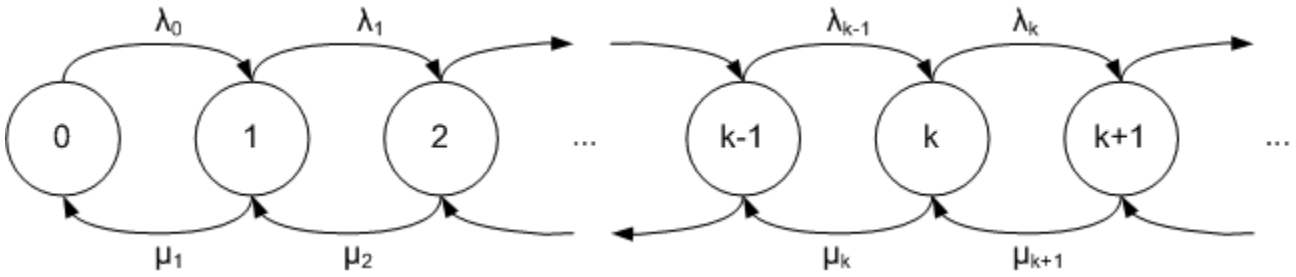
2.1.7 INFINITELY MANY SERVERS

While never exactly encountered in reality, an infinite-servers (e.g. **M/M/∞**) model is a convenient theoretical model for situations that involve storage or delay, such as parking lots, warehouses and even atomic transitions. In these models there is no queue, as such, instead each arriving customer receives service. When viewed from the outside, the model appears to delay or store each customer for some time.

2.2.0 **THE** BIRTH–**DEATH PROCESS**

The **birth–death process** is a special case of continuous-time Markov process where the states represent the current size of a population and where the transitions are limited to births and deaths. Birth–death processes have many applications in demography, queuing theory, performance engineering, biology, etc.

When a birth occurs, the process goes from state *n* to *n + 1*. When a death occurs, the process goes from state *n* to state *n − 1*. The process is specified by birth rates $\{\lambda_i\}_{i=0\ldots\infty}$ and death rates $\{\mu_i\}_{i=1\ldots\infty}$.



A **pure birth process** is a birth–death process where $\mu_i = 0$ *for all* $i \geq 0$.
A **pure death process** is a birth–death process where $\lambda_i = 0$ *for all* $i \geq 0$.
A (homogeneous) Poisson process is a pure birth process where $\lambda_i = \lambda$ for all $i \geq 0$

M/M/1 model and *M/M/c* model, both used in queuing theory, are birth–death processes used to describe customers in an infinite queue.

2.2.1 BIRTH AND DEATH PROCESS IN QUEUING THEORY

In queuing theory the birth–death process is the most fundamental example of a queuing model, the *M/M/C/K/∞/FIFO* (*in complete Kendall's notation*) queue. This is a queue with Poisson arrivals, drawn from an infinite population, and *C* servers with exponentially distributed service time with *K* places in the queue. Despite the assumption of an infinite population this model is a good model for various telecommunication systems.

2.2.2 M/M/1 queue

The M/M/1(Single Server Infinite Queue Model) is a single server queue with an infinite buffer size. In a non-random environment the birth–death process in queuing models tend to be long-term averages, so the average rate of arrival is given as λ and the average service time as 1/μ. The birth and death process is a M/M/1 queue when, $\lambda_i = \lambda$ and $\mu_i = \mu$ for all i.

The difference equations for the probability that the system is in state *k* at time *t* are,

$$p_0'(t) = \mu_1 p_1(t) - \lambda_0 p_0(t) \quad (1)$$

$$p_k'(t) = \lambda_{k-1} p_{k-1}(t) + \mu_{k+1} p_{k+1}(t) - (\lambda_k + \mu_k) p_k(t) \quad (2)$$

2.2.3 M/M/C queue

The M/M/C (Multiple Server Infinite Queue Model) is multi-server queue with *C* servers and an infinite buffer. This differs from the M/M/1 queue only in the service time which now becomes,

$\mu_i = i\mu$ for $i \leq C$ and $\mu_i = C\mu$ for $i \geq C$ with $\lambda_i = \lambda$ for all i.

2.2.4 M/M/1/K queue

The M/M/1/K (Single Server finite Queue Model) queue is a single server queue with a buffer of size *K*. This queue has applications in telecommunications, as well as in biology when a population has a capacity limit. In telecommunication we again use the parameters from the M/M/1 queue with,

$\lambda_i = \lambda$ for $0 \leq i < K$

$\lambda_i = 0$ for $i \geq K$

$\mu_i = \mu$ for $1 \leq i \leq K$.

For example, in biology, particularly the growth of bacteria, when the population is zero there is no ability to grow so,

$$\lambda_0 = 0.$$

Additionally if the capacity represents a limit where the population dies from over population,

$\mu_K = 0$.

The differential equations for the probability that the system is in state *k* at time *t* are,

$$p_0'(t) = \mu_1 p_1(t) - \lambda_0 p_0(t) \quad (3)$$

$$p_k'(t) = \lambda_{k-1} p_{k-1}(t) + \mu_{k+1} p_{k+1}(t) - (\lambda_k + \mu_k) p_k(t) \text{ for } k \leq K \quad (4)$$

$$p_k'(t) = 0 \text{ for } k > K \quad (5)$$

2.2.5 EQUILIBRIUM

A queue is said to be in equilibrium if the limit $\lim_{t\to\infty} p_k(t)$ exists. For this to be the case, $p_k'(t)$ must be zero.

Using the M/M/1 queue as an example, the steady state (equilibrium) equations are,

$$\lambda_0 p_0(t) = \mu_1 p_1(t) \quad (6)$$

$$(\lambda_k + \mu_k) p_k(t) = \lambda_{k-1} p_{k-1}(t) + \mu_{k+1} p_{k+1}(t) \quad (7)$$

If $\lambda_k = \lambda$ and $\mu_k = \mu$ for all k(the homogenous case), this can be reduced to

$$\lambda p_k(t) = \mu p_{k+1}(t) \text{ for } k \geq 0. \quad (8)$$

2.2.6 LIMIT BEHAVIOUR

In a small time Δt, only three types of transitions are possible: one death, or one birth, or no birth nor death. If the rate of occurrences (per unit time) of births is $\lambda$and that for deaths isμ, then the probabilities of the above transitions are $\lambda\Delta t$, $\mu\Delta t$, and $1-(\lambda+\mu)\Delta t$ respectively. For a population process, "birth" is the transition towards increasing the population by 1 while "death" is the transition towards decreasing the population size by 1.

The Formulae for Computing Performance Measures of Queuing Systems are shown in the appendix C.

2.3.0 DECISION SUPPORT SYSTEMS OVERVIEW

Computerized decision support systems became practical with the development of minicomputers, timeshare operating systems and distributed computing. The history of the implementation of such systems begins in the mid-1960s.

In a technology field as diverse as DSS, chronicling history is neither neat nor linear. Different people perceive the field of Decision Support Systems from various vantage points and report different accounts of what happened and what was important (Power, 2003). As technology evolved new computerized decision support applications were developed and studied. Researchers used multiple frameworks to help build and understand these systems. Today one can organize the history of DSS into the five broad DSS categories explained in Power (2001), including: communications-driven, data-driven, document driven, knowledge-driven and model-driven decision support systems.

2.3.1 DEFINITION OF DECISION SUPPORT SYSTEMS

A Decision Support System (DSS) is an interactive computer-based system or subsystem intended to help decision makers use communications technologies, data, documents, knowledge and/or models to identify and solve problems, complete decision process tasks, and make decisions. Decision Support System is a general term for any computer application that enhances a person or group's ability to make decisions. Also, Decision Support Systems refers to an academic field of research that involves designing and studying Decision Support Systems in their context of use. In general, Decision Support Systems are a class of computerized information system that supports decision-making activities.

A Decision Support System (DSS) is a collection of integrated software applications and hardware that form the backbone of an organization's decision making process. Companies across all industries rely on decision support tools, techniques, and models to help them assess and resolve everyday business questions. The decision support system is data-driven, as the entire process feeds off of the collection and availability of data to analyze. (http://www.microstrategy.com/decision-support-system)

Keen (1978), defined a Decision Support System as a computer-based information system that supports business or organizational decision-making activities. DSSs serve the management, operations, and planning levels of an organization and help to make decisions, which may be rapidly changing and not easily specified in advance. DSSs include knowledge-based systems. A properly designed DSS is an interactive software-based system intended to help decision makers compile useful information from a combination of raw data, documents,

personal knowledge, or business models to identify and solve problems and make decisions. (Keen, 1978)

Decision Support Systems may also be viewed as a specific class of computerized information system that supports business and organizational decision-making activities. A properly designed DSS is an interactive software-based system intended to help decision makers compile useful information from raw data, documents, personal knowledge, and/or business models to identify and solve problems and make decisions. (www.informationbuilders.com/...)

A Decision Support System may also be defined as a computer program application that analyzes business data and presents it so that users can make business decisions more easily. It is an "informational application" (to distinguish it from an "operational application" that collects the data in the course of normal business operation). A decision support system may present information graphically and may include an expert system or artificial intelligence (AI). It may be aimed at business executives or some other group of knowledge workers. (http://searchcio.techtarget.com/definition/decision-support-system)

2.3.2 HISTORY AND ORIGIN OF DECISION SUPPORT SYSTEMS

In the 1960s, researchers began systematically studying the use of computerized quantitative models to assist in decision making and planning (Raymond, 1966). Ferguson and Jones (1969) reported the first experimental study using a computer aided decision system. They investigated a production scheduling application running on an IBM 7094. In retrospect, a major historical turning point was ScottMorton's (1967) dissertation field research at Harvard University.

Scott Morton’s study involved building, implementing and then testing an interactive, model-driven management decision system. Fellow Harvard Ph.D. student Andrew McCosh asserts that the “concept of decision support systems was first articulated by Scott Morton in February 1964 in a basement office in Sherman Hall, Harvard Business School” (McCosh, 2002) in a discussion they had about Scott Morton’s dissertation. During 1966, Scott Morton (1971) studied how computers and analytical models could help managers make a recurring key business planning decision. He conducted an experiment in which managers actually used a Management Decision System (MDS). Marketing and production managers used an MDS to

coordinate production planning for laundry equipment. The MDS ran on an IDI 21 inch CRT with a light pen connected using a 2400 bps modem to a pair of Univac 494 systems.

The pioneering work of George Dantzig, Douglas Engelbart and Jay Forrester likely influenced the feasibility of building computerized decision support systems. In 1952, Dantzig became a research mathematician at the Rand Corporation, where he began implementing linear programming on its experimental computers. In the mid-1960s, Engelbart and colleagues developed the first hypermedia—groupware system called NLS (oNLine System). NLS facilitated the creation of digital libraries and the storage and retrieval of electronic documents using hypertext. NLS also provided for on-screen video teleconferencing and was a forerunner to group decision support systems. Forrester was involved in building the SAGE (Semi-Automatic Ground Environment) air defense system for North America completed in 1962. SAGE is probably the first computerized data-driven DSS. Also, Professor Forrester started the System Dynamics Group at the Massachusetts Institute of Technology Sloan School. His work on corporate modeling led to programming DYNAMO, a general simulation compiler.

In 1960, J.C.R. Licklider published his ideas about the future role of multi-access interactive computing in a paper titled "Man-Computer Symbiosis." He saw man-computer interaction as enhancing both the quality and efficiency of human problem solving and his paper provided a guide for decades of computer research to follow. Licklider was the architect of Project MAC at MIT that furthered the study of interactive computing.

By April 1964, the development of the IBM System 360 and other more powerful mainframe systems made it practical and cost-effective to develop Management Information Systems (MIS) for large companies (Davis, 1974). These early MIS focused on providing managers with structured, periodic reports and the information was primarily from accounting and transaction processing systems, but the systems did not provide interactive support to assist managers in decision making.

Around 1970 business journals started to publish articles on management decision systems, strategic planning systems and decision support systems. For example, Scott Morton and colleagues McCosh and Stephens published decision support related articles in 1968. The first use of the term decision support system was in Gorry and Scott-Morton's (1971) Sloan Management Review article. They argued that Management Information Systems primarily

focused on structured decisions and suggested that the supporting information systems for semi-structured and unstructured decisions should be termed “Decision Support Systems”.

Gerrity, Jr.(1971) focused on Decision Support Systems design issues in his 1971 Sloan Management Review article titled "The Design of Man-Machine Decision Systems: An Application to Portfolio Management". The article was based on his MIT Ph.D. dissertation. His system was designed to support investment managers in their daily administration of a clients' stock portfolio.

John D.C. Little, also at Massachusetts Institute of Technology, was studying DSS for marketing. Little and Lodish (1969) reported research on MEDIAC, a media planning support system. Also, Little (1970) identified criteria for designing models and systems to support management decision-making. His four criteria included: robustness, ease of control, simplicity, and completeness of relevant detail. All four criteria remain relevant in evaluating modern Decision Support Systems. By 1975, Little was expanding the frontiers of computer-supported modeling. His DSS called Brandaid was designed to support product, promotion, pricing and advertising decisions. Little also helped develop the financial and marketing modeling language known as EXPRESS.

In 1974, Gordon Davis, a Professor at the University of Minnesota, published his influential text on Management Information Systems. He defined a Management Information System as "an integrated, man/machine system for providing information to support the operations, management, and decision-making functions in an organization. (p. 5)." Davis's Chapter 12 was titled "Information System Support for Decision Making" and Chapter 13 was titled "Information System Support for Planning and Control". Davis’s framework incorporated computerized decision support systems into the emerging field of management information systems.

Peter Keen and Charles Stabell, (1978) claimed the concept of decision support systems evolved from "the theoretical studies of organizational decision making done at the Carnegie Institute of Technology during the late 1950s and early '60s and the technical work on interactive computer systems, mainly carried out at the Massachusetts Institute of Technology in the 1960s.” (Keen and Morton, 1978)

Hans Klein and Leif Methlie (1995,) noted “A study of the origin of DSS has still to be written. It seems that the first DSS papers were published by PhD students or professors in

business schools, who had access to the first time-sharing computer system: Project MAC at the Sloan School, the Dartmouth Time Sharing Systems at the Tuck School. In France, HEC was the first French business school to have a time-sharing system (installed in 1967), and the first DSS papers were published by professors of the School in 1970."

2.4.0 DECISION SUPPORT SYSTEMS THEORY DEVELOPMENT

Keen and ScottMorton's (1978) DSS work provided the first broad behavioral orientation to decision support system analysis, design, implementation, evaluation and development. This influential text provided a framework for teaching DSS in business schools. McCosh and Scott-Morton's (1978) DSS book was more influential in Europe.

Alter (1980) published his MIT doctoral dissertation results in an influential book. Alter's research and papers (1977) expanded the framework for thinking about business and management DSS. Also, his case studies provided a firm descriptive foundation of decision support system examples. A number of other MIT dissertations completed in the late 1970s also dealt with issues related to using models for decision support.

Alter concluded from his research (1980) that decision support systems could be categorized in terms of the generic operations that can be performed by such systems. These generic operations extend along a single dimension, ranging from extremely data-oriented to extremely model-oriented. Alter conducted a field study of 56 DSS that he categorized into seven distinct types of DSS. His seven types include:

- **File Drawer Systems**: These are the simplest type of DSS which can provide access to data items that are used to make a decision. A common example is the ATM Machine, which makes use of the balance of a customer's account to make transfer of funds decisions.
- **Data Analysis Systems**: These systems support access to data and the manipulation of such data by computerized tools tailored to a specific task and setting or by more general tools and operators. Example, the Airline Reservation system which uses information to make flight plans.

- **Analysis Information Systems:** provide access to a series of decision-oriented databases and small models. In this system the information from one file, table, can be combined with information from other files to answer a specific query.
- **Accounting and Financial Models:** these systems use internal accounting data and accounting modeling capabilities to calculate production cost and to make pricing decisions. Such systems cannot handle uncertainty.
- **Representational Models** are systems that estimate the consequences of actions on the basis of simulation models. They can incorporate uncertainty and make use of models to solve decision problem using forecasts. They can be used to augment the capabilities of Accounting models
- **Optimization Models: are systems** that provide guidelines for action by generating an optimal solution consistent with a series of constraints. They are used to estimate the effects of different decision alternative and incorporate uncertainty in decision making.
- **Suggestion Models**: refer to systems that perform the logical processing leading to a specific suggested decision for a fairly structured or well-understood task.

Donovan and Madnick (1977) classified DSS as institutional or ad hoc. Institutional DSS support decisions that are recurring. An ad hoc DSS supports querying data for one time requests. Hackathorn and Keen (1981) identified DSS in three distinct yet interrelated categories: Personal DSS, Group DSS and Organizational DSS.

In 1979, John Rockart of the Harvard Business School published a ground breaking article that led to the development of executive information systems (EISs) or executive support systems (ESS). Rockart developed the concept of using information systems to display critical success metrics for managers.

Robert Bonczek, Clyde Holsapple, and Andrew Whinston (1981) explained a theoretical framework for understanding the issues associated with designing knowledge-oriented Decision Support Systems. They identified four essential "aspects" or general components that were common to all DSS:

1. A **Language System (LS)** that specifies all messages a specific DSS can accept
2. A **Presentation System (PS**) for all messages a DSS can emit;
3. **A Knowledge System (KS)** for all knowledge a DSS has; and

4. **A Problem-Processing System (PPS)** that is the "software engine" that tries to recognize and solve problems during the use of a specific DSS. Their book explained how Artificial Intelligence and Expert Systems technologies were relevant to developing DSS.

Finally, Ralph Sprague and Eric Carlson's (1982) book Building Effective Decision Support Systems was an important milestone. Much of the book further explained the Sprague (1980) DSS framework of data base, model base and dialog generation and management software. Also, it provided a practical and understandable overview of how organizations could and should build DSS. Sprague and Carlson (1982) defined DSS as "a class of information system that draws on transaction processing systems and interacts with the other parts of the overall information system to support the decision-making activities of managers and other knowledge workers in organizations."

2.5.0 **DECISION SUPPORT SYSTEM APPLICATIONS DEVELOPMENT**

Beginning in about 1980 many activities associated with building and studying DSS occurred in universities and organizations that resulted in expanding the scope of DSS applications. These actions also expanded the field of decision support systems beyond the initial business and management application domain. These diverse systems were all called Decision Support Systems. From those early days, it was recognized that DSS could be designed to support decision-makers at any level in an organization. Also, DSS could support operations decision making, financial management and strategic decision-making.

A literature survey and citation studies (Alavi and Joachimsthaler, 1990) suggest the major applications for DSS emphasized manipulating quantitative models, accessing and analyzing large data bases, and supporting group decision making. Much of the model-driven DSS research emphasized use of the systems by individuals, i.e., personal DSS, while data-driven DSS were usually institutional, ad hoc or organizational DSS. Group DSS research emphasized impacts on decision process structuring and especially brainstorming.

2.5.1 **Model-driven DSS**

Scott-Morton's (1971) production planning management decision system was the first widely discussed model-driven DSS, but Ferguson and Jones' (1969) production scheduling application was also a model-driven DSS.

A model-driven DSS emphasizes access to and manipulation of financial, optimization and/or simulation models. Simple quantitative models provide the most elementary level of functionality. Model-driven DSS use limited data and parameters provided by decision makers to aid decision makers in analyzing a situation, but in general large data bases are not needed for model-driven DSS (Power, 2002). Early versions of model-driven DSS were called model-oriented DSS by Alter (1980), computationally oriented DSS by Bonczek, Holsapple and Whinston (1981) and later spreadsheet-oriented and solver-oriented DSS by Holsapple and Whinston (1996).

The first commercial tool for building model-driven DSS using financial and quantitative models was called IFPS, an acronym for interactive financial planning system. It was developed in the late 1970's by Gerald R. Wagner and his students at the University of Texas. Wagner's company, EXECUCOM Systems, marketed IFPS until the mid 1990s. Gray's Guide to IFPS (1983) promoted the use of the system in business schools. Another DSS generator for building specific systems based upon the Analytic Hierarchy Process (Saaty, 1982), called Expert Choice, was released in 1983. Expert Choice supports personal or group decision making. Ernest Forman worked closely with Thomas Saaty to design Expert Choice.

In 1978, Dan Bricklin and Bob Frankston co-invented the software program VisiCalc (Visible Calculator). VisiCalc provided managers the opportunity for hands-on computer-based analysis and decision support at a reasonably low cost. VisiCalc was the first "killer" application for personal computers and made possible development of many model-oriented, personal DSS for use by managers. The history of microcomputer spreadsheets is described in Power (2000). In 1987, Frontline Systems founded by Dan Fylstra marketed the first optimization solver add-in for Microsoft Excel.

In a 1988 paper, Sharda, Barr, and McDonnell reviewed the first 15 years of model-driven DSS research. They concluded that research related to using models and financial planning systems for decision support was encouraging but certainly not uniformly positive. As computerized models became more numerous, research focused on model management and on enhancing more diverse types of models for use in DSS such as multi-criteria, optimization and simulation models.

The idea of model-driven spatial decision support system (SDSS) evolved in the late 1980's (Armstrong, et al., 1986) and by 1995 the SDSS concept had become firmly established in the literature (Crossland, et al., 1995). Data-driven spatial DSS are also common.

2.5.2 **Data-Driven DSS**

In general, a data-driven DSS emphasizes access to and manipulation of a time-series of internal company data and sometimes external and real-time data. Simple file systems accessed by query and retrieval tools provide the most elementary level of functionality. Data warehouse systems that allow the manipulation of data by computerized tools tailored to a specific task and setting or by more general tools and operators provide additional functionality. Data-Driven DSS with On-line Analytical Processing (Codd et al., 1993) provide the highest level of functionality and decision support that is linked to analysis of large collections of historical data. Executive Information Systems are examples of data-driven DSS (Power, 2002). Initial examples of these systems were called data-oriented DSS, Analysis Information Systems (Alter, 1980) and retrieval-only DSS by Bonczek, Holsapple and Whinston (1981).

One of the first data-driven DSS was built using an APL-based software package called AAIMS, An Analytical Information Management System. It was developed from 1970-1974 by Richard Klaas and Charles Weiss at American Airlines (Alter, 1980).

As noted previously, in 1979 John Rockart's research stimulated the development of executive information systems (EIS) and executive support systems (ESS). These systems evolved from single user model-driven decision support systems and from the development of relational database products. The first EIS used pre-defined information screens maintained by analysts for senior executives. For example, in the Fall of 1978, development of an EIS called Management Information and Decision Support (MIDS) system began at Lockheed-Georgia (Houdeshel and Watson, 1987).

The first EIS were developed in the late 1970s by Northwest Industries and Lockheed "who risked being on the 'bleeding edge' of technology …. Few even knew about the existence of EIS until John Rockart and Michael Treacy's article, 'The CEO Goes On-line,' appeared in the January-February 1982 issue of the Harvard Business Review. (Watson, et al, 1997, p. 6)" Watson and colleagues further note, "A major

contributor to the growth of EIS was the appearance of vendor-supplied EIS software in the mid-1980s. Pilot Software's Command Center and Comshare's Commander EIS made it much easier for firms to develop an EIS by providing capabilities for (relatively) easy screen design, data importation, user-friendly front ends, and access to news services. (p. 6)" In a related development in 1984, Teradata's parallel processing relational database management system shipped to customers Wells Fargo and ATand T.

In about 1990, data warehousing and On-Line Analytical Processing (OLAP) began broadening the realm of EIS and defined a broader category of data-driven DSS (Dhar and Stein, 1997). Nigel Pendse (1997), author of the OLAP Report, claims both multidimensional analysis and OLAP had origins in the APL programming language and in systems like Express and Comshare's System W. Nylund (1999) traces the developments associated with Business Intelligence (BI) to Procter and Gamble's efforts in 1985 to build a DSS that linked sales information and retail scanner data. Metaphor Computer Systems, founded by researchers like Ralph Kimball from Xerox's Palo Alto Research Center (PARC), built the early Pand G data-driven DSS. Staff from Metaphor later founded many of the Business Intelligence vendors: The term BI is a popularized, umbrella term coined and promoted by Howard Dresner of the Gartner Group in 1989. It describes a set of concepts and methods to improve business decision making by using fact-based support systems. BI is sometimes used interchangeably with briefing books, report and query tools and executive information systems. In general, business intelligence systems are data-driven DSS.

Bill Inmon and Ralph Kimball actively promoted decision support systems built using relational database technologies. Barry Devlin (Devlin and Murphy, 1988) defined IBM's data warehouse architecture and promoted it. For many Information Systems practitioners, DSS built using Oracle or DB2 were the first decision support systems they read about in the popular computing literature. Ralph Kimball was "The Doctor of DSS" and Bill Inmon was the "father of the data warehouse". By 1995, Wal-Mart's data-driven DSS had more than 5 terabytes of on-line storage from Teradata that expanded to more than 24 terabytes in 1997. In more recent years, vendors added tools to create web-based dashboards and scorecards.

2.5.3 **Communications-Driven DSS**

Communications-driven DSS use network and communications technologies to facilitate decision-relevant collaboration and communication. In these systems, communication technologies are the dominant architectural component. Tools used include groupware, video conferencing and computer-based bulletin boards (Power, 2002).

Engelbart's 1962 paper "Augmenting Human Intellect: A Conceptual Framework" is the anchor for much of the later work related to communications-driven DSS. In 1969, he demonstrated the first hypermedia/groupware system NLS (oNLine System) at the Fall Joint Computer Conference in San Francisco. Engelbart invented the both the computer mouse and groupware.

Joyner and Tunstall's article (1970) reporting testing of their Conference Coordinator computer software is the first empirical study in this research area. Turoff's (1970) article introduced the concept of Computerized Conferencing. He developed and implemented the first Computer Mediated Communications System (EMISARI) tailored to facilitate group communications.

In the early 1980s, academic researchers developed a new category of software to support group decision-making called Group Decision Support Systems abbreviated GDSS (Gray, 1981). Mindsight from Execucom Systems, GroupSystems developed at the University of Arizona and the SAMM system developed by University of Minnesota researchers were early Group DSS.

Eventually GroupSystems matured into a commercial product. Jay Nunamaker, Jr. and his colleagues wrote in 1992 that the underlying concept for GroupSystems had its beginning in 1965 with the development of Problem Statement Language/Problem Statement Analyzer at Case Institute of Technology. In 1984, the forerunner to GroupSystems called PLEXSYS was completed and a computer-assisted group meeting facility was constructed at the University of Arizona. The first Arizona facility, called the PlexCenter, housed a large U-shaped conference table with 16 computer workstations.

On the origins of SAMM, Dickson, Poole and DeSanctis (1992) report that Brent Gallupe, a Ph.D. student at the University of Minnesota, decided in 1984 "to program his own small GDSS system in BASIC and run it on his university's VAX computer".

DeSanctis and Gallup (1987) defined two types of GDSS. Basic or level 1 GDSS are systems with tools to reduce communication barriers, such as large screens for display of ideas, voting mechanisms, and anonymous input of ideas and preferences. These are communications-

driven DSS. Advanced or level 2 GDSS provide problem-structuring techniques, such as planning and modeling tools. These are model-driven group DSS. Since the mid-1980s, many research studies have examined the impacts and consequences of both types of group DSS. Also, companies have commercialized model-driven group DSS and groupware.

Kersten (1985) developed NEGO, a computerized group tool to support negotiations. Bui and Jarke (1986) reported developing Co-op, a system for cooperative multiple criteria group decision support. Kraemer and King (1988) introduced the concept of Collaborative Decision Support Systems (CDSSs). They defined them as interactive computer-based systems to facilitate the solution of ill-structured problems by a set of decision makers working together as a team.

In 1989, Lotus introduced a groupware product called Notes and broadened the focus of GDSS to include enhancing communication, collaboration and coordination among groups of people. Notes had its roots in a product called PLATO Notes, written at the Computer-based Education Research Laboratory (CERL) at the University of Illinois in 1973 by David R. Woolley.

In general, groupware, bulletin boards, audio and videoconferencing are the primary technologies for communications-driven decision support. In the past few years, voice and video delivered using the Internet protocol have greatly expanded the possibilities for synchronous communications-driven DSS.

Document-driven DSS uses computer storage and processing technologies to provide document retrieval and analysis. Large document databases may include scanned documents, hypertext documents, images, sounds and video. Examples of documents that might be accessed by a document-driven DSS are policies and procedures, product specifications, catalogs, and corporate historical documents, including minutes of meetings and correspondence. A search engine is a primary decision-aiding tool associated with a document-driven DSS (Power, 2002). These systems have also been called text-oriented DSS (Holsapple and Whinston, 1996).

The precursor for this type of DSS is Vannevar Bush's (1945) article titled "As We May Think". Bush wrote "Consider a future device for individual use, which is a sort of mechanized private file and library. It needs a name, and to coin one at random, 'memex' will do". Bush's memex is a much broader vision than that of today's document-driven DSS.

Text and document management emerged in the 1970s and 1980s as an important, widely used computerized means for representing and processing pieces of text (Holsapple and

Whinston, 1996). The first scholarly article for this category of DSS was written by Swanson and Culnan (1978). They reviewed document-based systems for management planning and control. Until the mid-1990s little progress was made in helping managers find documents to support their decision making. Fedorowicz (1993) helped define the need for such systems. She estimated in her 1996 article that only 5 to 10 percent of stored business documents are available to managers for use in decision making. The World-wide web technologies significantly increased the availability of documents and facilitated the development of document-driven DSS.

2.5.4 **Knowledge-driven DSS**

Knowledge-driven DSS can suggest or recommend actions to managers. These DSS are person-computer systems with specialized problem-solving expertise. The "expertise" consists of knowledge about a particular domain, understanding of problems within that domain, and "skill" at solving some of these problems (Power, 2002). These systems have been called suggestion DSS (Alter, 1980) and knowledge-based DSS (Klein and Methlie, 1995). Goul, Henderson, and Tonge (1992) examined Artificial Intelligence (AI) contributions to DSS.

In 1965, a Stanford University research team led by Edward Feigenbaum created the DENDRAL expert system. DENDRAL led to the development of other rule-based reasoning programs including MYCIN, which helped physicians diagnose blood diseases based on sets of clinical symptoms. The MYCIN project resulted in development of the first expert-system shell (Buchanan and Shortliffe, 1984).

Bonczek, Holsapple and Whinston's (1981) book created interest in using these technologies for DSS. In 1983, Dustin Huntington established EXSYS. That company and product made it practical to use PC based tools to develop expert systems. By 1992, some 11 shell programs were available for the MacIntosh platform, 29 for IBM-DOS platforms, 4 for Unix platforms, and 12 for dedicated mainframe applications (National Research Council, 1999). Artificial Intelligence systems have been developed to detect fraud and expedite financial transactions, many additional medical diagnostic systems have been based on AI, expert systems have been used for scheduling in manufacturing operation and web-based advisory systems. In recent years, connecting expert systems technologies to relational databases with web-based front ends has broadened the deployment and use of knowledge-driven DSS.

2.5.5 **Web-Based DSS**

Beginning in approximately 1995, the World-wide Web and global Internet provided a technology platform for further extending the capabilities and deployment of computerized decision support. The release of the HTML 2.0 specifications with form tags and tables was a turning point in the development of web-based DSS. In 1995, a number of papers were presented on using the Web and Internet for decision support at the 3rd International Conference of the International Society for Decision Support Systems (ISDSS). In addition to Web-based, model-driven DSS, researchers were reporting Web access to data warehouses. DSS Research Resources was started as a web-based collection of bookmarks. By 1995, the World-Wide Web (Berners-Lee, 1996) was recognized by a number of software developers and academics as a serious platform for implementing all types of Decision Support Systems (Bhargava and Power, 2001).

In November 1995, Power, Bhargava and Quek submitted the Decision Support Systems Research page for inclusion in ISWorld. The goal was to provide a useful starting point for accessing Web-based material related to the design, development, evaluation, and implementation of Decision Support Systems. Nine months later, a DSS/WWW Workshop organized by Power and Quek was held as part of the IFIP Working Group 8.3 Conference on "Implementing Systems for Supporting Management Decisions: Concepts, Methods and Experiences", July 21-24, 1996 in London, UK.

In 1996-97, corporate intranets were developed to support information exchange and knowledge management. The primary decision support tools included ad hoc query and reporting tools, optimization and simulation models, online analytical processing (OLAP), data mining and data visualization (Powell, 2001). Enterprise-wide DSS using database technologies were especially popular in Fortune 2000 companies (Power, 1997). Bhargava, et al (1997) continued to discuss and experiment with electronic markets for decision technologies.

In 1999, vendors introduced new Web-based analytical applications. Many DBMS vendors shifted their focus to Web-based analytical applications and business intelligence solutions. In 2000, application service providers (ASPs) began hosting the application software and technical infrastructure for decision support capabilities. 2000 was also the year of the portal. More sophisticated "enterprise knowledge portals" were introduced by vendors that

combined information portals, knowledge management, business intelligence, and communications-driven DSS in an integrated Web environment (Bhargava and Power, 2001).

Power (1998) defined a Web-based decision support system as a computerized system that delivers decision support information or decision support tools to a manager or business analyst using a "thin-client" Web browser like Netscape Navigator or Internet Explorer. The computer server that is hosting the DSS application is linked to the user's computer by a network with the TCP/IP protocol.

2.6 DECISION SUPPORT SYSTEMS DEVELOPMENT FRAMEWORKS

"DSS systems are not entirely different from other systems and require a structured approach. Such a framework includes people, technology, and the development approach." (Sprague and Carlson, 1982)

DSS technology levels (of hardware and software) may include:

1. The actual application that will be used by the user. This is the part of the application that allows the decision maker to make decisions in a particular problem area. The user can act upon that particular problem.
2. Generator contains Hardware/software environment that allows people to easily develop specific DSS applications. This level makes use of case tools or systems such as Crystal, AIMMS, and iThink.
3. Tools include lower level hardware/software. DSS generators including special languages, function libraries and linking modules

An iterative developmental approach allows for the DSS to be changed and redesigned at various intervals. Once the system is designed, it will need to be tested and revised for the desired performance.

2.7 CLASSIFICATION OF DECISION SUPPORT SYSTEMS

There are several ways to classify DSS applications. Not every DSS fits neatly into one of the categories, but may be a mix of two or more architectures.

Holsapple and Whinston (1996) classified DSS into the following six frameworks:

- Text-oriented DSS
- Database-oriented DSS

- Spreadsheet-oriented DSS
- Solver-oriented DSS
- Rule-oriented DSS
- Compound DSS.

A compound DSS is the most popular classification for a DSS. It is a hybrid system that includes two or more of the five basic structures described by Holsapple and Whinston. The support given by DSS can be separated into three distinct, interrelated categories (Hackathorn and Keen, 1981):

- Personal Support,
- Group Support, and
- Organizational Support.

DSS components may be classified as:

1. **Inputs:** Factors, numbers, and characteristics to analyze
2. **User Knowledge and Expertise:** Inputs requiring manual analysis by the user
3. **Outputs:** Transformed data from which DSS "decisions" are generated
4. **Decisions:** Results generated by the DSS based on user criteria

Decision Support Systems which perform selected cognitive decision-making functions and are based on artificial intelligence or intelligent agents technologies are called Intelligent Decision Support Systems (IDSS).

The nascent field of Decision engineering treats the decision itself as an engineered object, and applies engineering principles such as Design and Quality assurance to an explicit representation of the elements that make up a decision.

2.8 APPLICATIONS OF DECISION SUPPORT SYSTEMS

There are theoretical possibilities of building Decision support systems in any knowledge domain.

One example is the clinical decision support system for medical diagnosis. Other examples include a bank loan officer verifying the credit of a loan applicant or an engineering firm that has bids on several projects and wants to know if they can be competitive with their costs.

DSS is extensively used in business and management. Executive dashboard and other business performance software allow faster decision making, identification of negative trends, and better allocation of business resources.

A growing area of DSS application, concepts, principles, and techniques is in agricultural production, marketing for sustainable development. For example, the DSSAT4 package, developed through financial support of USAID during the 80's and 90's, has allowed rapid assessment of several agricultural production systems around the world to facilitate decision-making at the farm and policy levels. There are, however, many constraints to the successful adoption on DSS in agriculture. (http://www.icasa.net/dssat/index.html)

DSS are also prevalent in forest management where the long planning time frame demands specific requirements. All aspects of Forest management, from log transportation, harvest scheduling to sustainability and ecosystem protection have been addressed by modern DSSs.

A specific example concerns the Canadian National Railway system, which tests its equipment on a regular basis using a decision support system. A problem faced by any railroad is worn-out or defective rails, which can result in hundreds of derailments per year. Under a DSS, CN managed to decrease the incidence of derailments at the same time other companies were experiencing an increase.

2.9 **BENEFITS OF DECISION SUPPORT SYSTEMS**

- Improves personal efficiency
- Speed up the process of decision making
- Increases organizational control
- Encourages exploration and discovery on the part of the decision maker
- Speeds up problem solving in an organization
- Facilitates interpersonal communication
- Promotes learning or training
- Generates new evidence in support of a decision
- Creates a competitive advantage over competition
- Reveals new approaches to thinking about the problem space
- Helps automate managerial processes

2.10 THE CHARACTERISTICS OF IDEAL DECISION SUPPROT SYSTEMS

It is a generally acceptable standard that unified reporting, analytical, and monitoring platform form the core of any Decision Support System. Some of the important characteristics of an ideal Decision Support System are:

- **Support for individual and group decision making:** a good Decision Support System provides a single platform that allows all users to access the same information and access the same version of truth, while providing autonomy to individual users and development groups to design reporting content locally.
- **Ease of Development and Deployment:** an ideal DSS should deliver an interactive, scalable platform for rapidly developing and deploying projects. Multiple projects can be created within a single shared metadata. Within each project, development teams should be able to create a wide variety of re-usable metadata objects. As decision support system deployment expands within an organization, an ideal DSS platform should effortlessly support an increasing concurrent user base.
- **Comprehensive Data Access:** A well designed DSS should allow users to access data from different sources concurrently, leaving organizations the freedom to choose the data warehouse that best suits their unique requirements and preferences.
- **Integrated software:** An ideal DSS should provide an integrated platform that enables administrators and IT professionals to develop data models, perform sophisticated analysis, generate analytical reports, and deliver these reports to end users via different channels (Web, email, file, print and mobile devices). This eliminates the need for companies to spend countless effort purchasing and integrating disparate software products in an attempt to deliver a consistent user experience.
- **Flexibility:** a well designed DSS should provide an extensive library of APIs, so that customers can choose to leverage the power of the software's flexible APIs to design and deploy solutions tailored to their unique business needs. (www.microstrategy.com/decision-support-system/)

CHAPTER THREE

RESEARCH METHODOLOGY

3.0 INTRODUCTION

Research methodology is a way to systematically solve the research problem. It may be understood as a science of studying how research is done scientifically. In it we study the various steps that are generally adopted by a researcher in studying his research problem along with the logic behind them. (http://www.newagepublishers.com/samplechapter/000896.pdf)

This chapter describes the methods adopted in designing the decision support system for this project work. The obvious trend in coming up with a new software or technology is to undertake a careful and analytical study otherwise known as System Analysis. This is necessary as to avoid the risk of choosing and adopting an obsolete technology for the system under consideration.

According to Jeffrey, et al. (2004), System analysis is the study of problem domains to recommend improvements and specify the requirements of the solution. Therefore, System

Analysis concepts shall be exploited to unearth the methodical processes used in this study. This chapter is therefore discussed under the following sub-headings:

- System analysis
- Design objectives
- Research approach
- Research methodology
- File design and specifications
- File organization
- Programming Language used
- Program database
- System control
- System requirements

3.1 SYSTEM **ANALYSIS**

For best results and outcomes in the design of the decision support system for queue management, the Object Oriented Analysis (OOA) was adopted for system analysis.

The object oriented analysis techniques was used to study existing objects to see if they can be re-used or adopted for new use and also, to define new or modified objects that will be combined with existing objects into useful business applications. The technique is best suited for projects that will implement systems using emerging object technologies to construct, manage and assemble those objects into useful computer applications.

Object oriented analysis system development approaches have deliberately separated concerns of data and those of processes. OOA emerged to eliminate this artificial separation of concerns about data and process. Instead specific data and the processes that create, read, update and delete that data are integrated into a construct an object. The only way to read, update and delete that object's data is through (called properties) is through one of its embedded processes (called methods).

Object Oriented Analysis is a model driven technique that integrates data and processes concerns into constructs called objects. Object Oriented Analysis models are pictures that illustrate the systems objects from various perspectives such as structure and behavior.

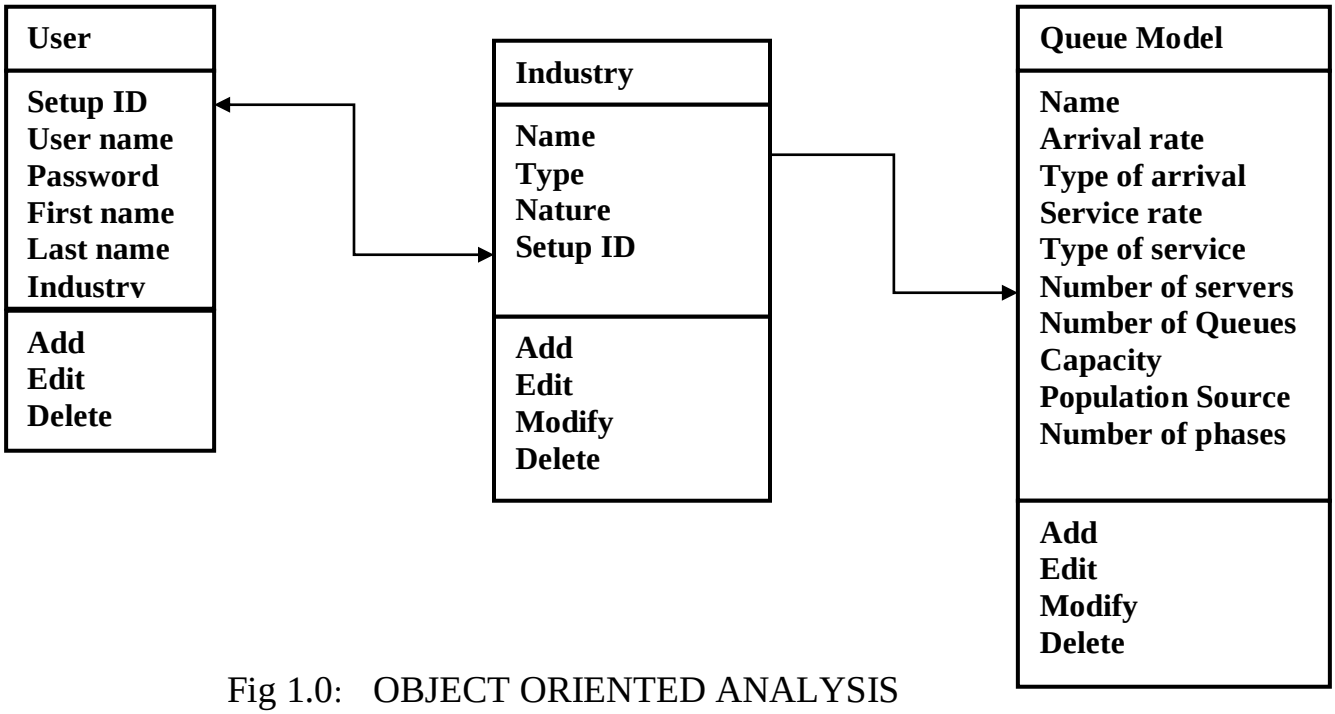


Fig 1.0: OBJECT ORIENTED ANALYSIS MODEL

3.2 **DESIGN OBJECTIVES**

Design Objective specifies a performance characteristic that may be used in the preparation of specifications for development or procurement of new equipment or systems. (http://en.wikipedia.org/wiki/Design_objective)

The design objective for this project is to develop a Decision support system for queue management with enablement for in-depth analysis of queuing situations and data management which is also cost effective, efficient and consistent to proffer solution to the problems associated with the management of queue situations.

The decision support system provides a Graphical User Interface for data entry by the User, a way to validate the entered data, and data storage in a database. The data input forms contain all of the vital parameters for the specified queue situation, including such items as arrival rate, service rate, number of servers, system capacity. The system is designed to analyze queuing models and display outputs on desired performance measures like server utilization,

average number of entities in queue, average number of entities in the system, average waiting time in queue, average waiting time in the system, etc.

3.3 **RESEARCH APPROACH**

According to Redman and Mory (1923), Research can either be applied (or action) research or fundamental (to basic or pure) research. Applied research aims at finding a solution for an immediate problem facing a society or an industrial/business organization, whereas fundamental research is mainly concerned with generalizations and with the formulation of a theory.

Applied research is a form of systematic inquiry involving the practical application of science. It accesses and uses some part of the research communities' (the academy's) accumulated theories, knowledge, methods, and techniques, for a specific, often state, business, or client driven purpose.

Applied research deals with solving practical problems and generally employs empirical methodologies. Because applied research resides in the messy real world, strict research protocols may need to be relaxed. For example, it may be impossible to use a random sample. Thus, transparency in the methodology is crucial. Implications for interpretation of results brought about by relaxing an otherwise strict canon of methodology should also be considered (Young, 1947).

Thus, the central aim of applied research is to discover a solution for some pressing practical problem, whereas basic research is directed towards finding information that has a broad base of applications and thus, adds to the already existing organized body of scientific knowledge.

In the light of the above definition, the applied research approach was adopted for the project work on the Design of Model-Driven Decision Support Platform for Queue Management.

3.4 **RESEARCH METHODOLOGY**

For this project work, Object Oriented Methodology (OOM) was employed. Object Oriented Methodology (OOM) is a systematical development approach which encourages and facilitates the re-use of software components. This methodology is a powerful tool in modeling and structuring complex software systems and allows the developer to deal with the same or very similar abstractions and entities during the complete process of software development.

The use of the Object Oriented Methodology provides better quality, higher productivity, and a low maintenance cost.

Object-Oriented development requires that object-oriented techniques be used during the analysis, and implementation of the system. This methodology asks the analyst to determine what the objects of the system are, how they behave over time or in response to events, and what responsibilities and relationships an object has to other objects. Object-oriented analysis has the analyst look at all the objects in a system, their commonalties, difference, and how the system needs to manipulate the objects. The Object Oriented Methodology of Building Systems takes the objects as the basis. For this, first the system to be developed is observed and analyzed and the requirements are defined as in any other method of system development. Once this is done, the objects in the required system are identified. For example in case of a Banking System, a customer is an object, a cheque book is an object, and even an account is an object.

In simple terms, Object Modeling is based on identifying the objects in a system and their interrelationships. Once this is done, the coding of the system is done. Object Modeling is somewhat similar to the traditional approach of system designing, in that it also follows a sequential process of system designing but with a different approach. (www.freetutes.com/systemanalysis/sa2-object-oriented-methodology.html)

3.5.0 **PROGRAM DESIGN**

The program or software design uses structured programming principle with top-down approach which breaks the program or software into modular forms.

The Model driven decision support platform for queue management contains the following modules:

I. User login/ Authentication module
II. Application Setup module
III. Industry Setup Module
IV. User Setup Module
V. Analysis/ tools module

3.5.1 **USER LOGIN/ AUTHENTICATION MODULE**

This module allows a user of the decision support platform to gain access into the system. This module serves as a system security device to guard against the dangers of unauthorized

access into the system and possible compromise of the system configurations and data. The module also helps in setting up the system for analysis of specified queuing systems as each user of the decision support system is tied to a specific queue system configuration for ease of analysis and data storage in the system's database.

3.5.2 APPLICATION SETUP MODULE

This is easily the most important module of the decision support system. The module comprise of two main sub-modules namely: **The User Account Setup module** and **the Industry Setup Module.**

3.5.3 THE USER ACCOUNT SETUP MODULE

The module handles the tasks of setting up the vital attributes of Users of the system by providing capabilities for adding, updating and deleting user information. The module also keeps track of the specific user's preferred configuration setup and industry. The module accepts the following as inputs: User Name, User Password, User Full Names, Setup Identification Number, and User Industry choice.

3.5.4 THE INDUSTRY/ APPLICATION SETUP MODULE

This module takes care of the task of configuring the decision support system to reflect the queuing characteristics that best represent a specific user's queuing situation and industry. The inputs of this module form the basis of the main variables which enable the decision support system to determine the type of queue model applicable to the specified queuing situation and industry.

The module accepts the following variables as inputs: Setup ID, Industry Type, Number of Servers, Nature of Queues, Numbers of Phases, Nature of service, Population Source.

3.6 QUEUE ANALYSIS MODULE

This module is used for the analysis of the various queue models that best represent the unique queuing situations of specific users of the decision support system. The module comprises of sub- modules for General Analysis, Sensitivity Analysis, Cost related analysis and Optimality analysis.

3.7.0 FILE DESIGN AND SPECIFICATIONS

The decision support system is designed to be able to process the data supplied by a user in real time and/or, previously stored data, to simplify the process of decision. In other words, the model driven decision support system is essentially a data centric application that stores, processes and outputs the results according to some set out criteria. Therefore, each of the data fields that make up decision support system software platform has different characteristics in respect to type, name and length. This section defines each data field used in the database in order to keep at check error due to incorrect entry of data fields and their associated problems. This was done following each of the modules of the program as follows:

3.7.1 **AUTHENTICATION MODULE DATABASE FILE DESIGN**

Field Name	**User Name**	**Password**
Field Type	nvarchar	nvarchar
Field Length	50	50

3.7.2 **THE USER ACCOUNT SETUP MODULE FILE DESIGN**

Field Name	**Setup ID**	**UserName**	Password	Full Names	Industry
Field Type	Integer	nvarchar	nvarchar	nvarchar	nvarchar
Field Length	25	50	50	50	50

3.7.3 **THE INDUSTRY/ APPLICATION SETUP MODULE FILE DESIGN**

Field Name	**Setup ID**	**Industry**	**Number of Severs**	**Nature of Queues**	**Nature of Service**	**Number of Phases**	**Population source**
Field Type	*int*	*nvarchar*	*nvarchar*	*nvarchar*	*nvarchar*	*nvarchar*	*nvarchar*
Field length	20	50	50	50	50	50	50

3.7.4 **FILE ORGANIZATION**

Random file organization is an organization in which records can be accessed randomly with the help of some key. The correspondence is maintained by direct address indexing or key indexing which helps the process of accessing any record in a database directly based on an address or key specified in the index of the particular record.

Random file organization was used to store data fields into the database. This was done based on the fact that access can be gained randomly with the programming language and the database used in the software design.

3.8.0 **QUEUE MANAGEMENT SYSTEM DESIGN TASKS**

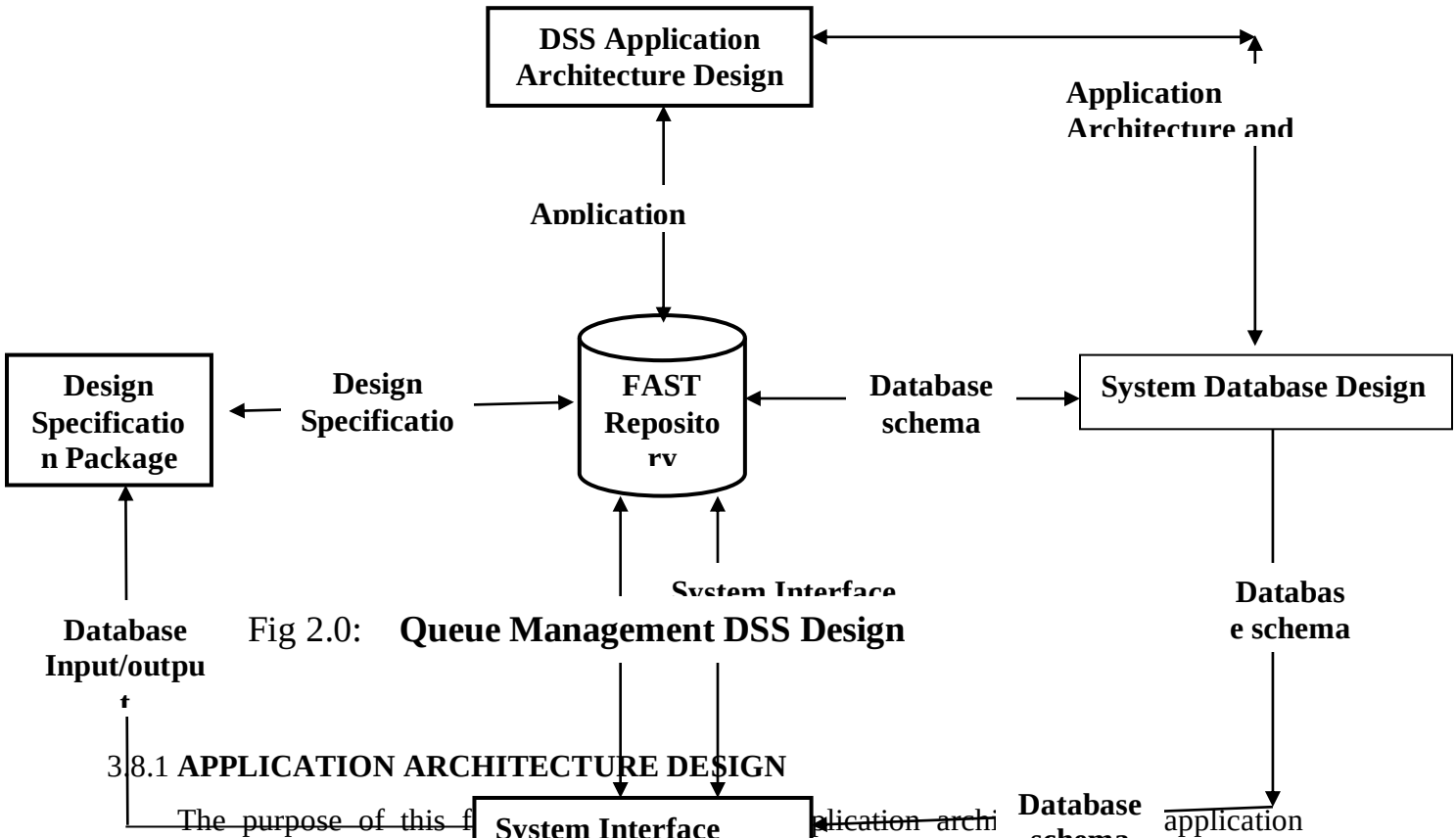


Fig 2.0: **Queue Management DSS Design**

3.8.1 **APPLICATION ARCHITECTURE DESIGN**

The purpose of this f[illegible]lication archi[illegible] application architecture defines the technologies to be used by (and used to build) one or more or all

Information systems in terms of its data, processes, interfaces and network components. It involves considering network technologies and making decisions on how the system's data processes and interfaces are to be distributed among business locations.

Application architecture design is accomplished by analyzing the data models and process models that were initially created during requirement analysis.

3.8.2 SYSTEM DATABASE DESIGN

The design of databases goes beyond the simple layout of records. Databases are shared resources and since many programs will typically use them in the future, in designing the database for this project attention was paid to designing databases that are adaptable to future requirements in order to improve the performance of the designed system. Special attention was paid to the issues of programming data structures, their impact on flexibility and performance, issues of record size and storage volume requirements. The aim is to prepare a technical design specification for a database that will be adaptable to future requirements and expansion.

A key input to the process of database design is application architecture and distribution analysis from prior design tasks.

3.8.3 SYSTEM INTERFACE DESIGN

After the database design, the next task was the development of Input, output, and dialogue specifications. To accomplish this, special care was taken to ensure precise layout and format of outputs interfaces. Internal controls were carefully specified to ensure that outputs are not lost, misrouted, misused or incomplete.

The data capture used was a designed windows form equipped with the necessary control objects to make it easy for the data to be recorded in the form. Finally, editing controls and error exception checkers were included to ensure data input accuracy.

3.8.4 THE USER INTERFACE DESIGN

The user interface establishes the dialogue between the system user and the computer. The dialogue determines everything from starting the decision support system, login into the system, to the tasks of setting up options and preferences, up to the task of getting help. The user interface functions to ensure the presentation of outputs and the screen to screen

transitions that can occur. The user has many options through menu strip items to make the system accommodating and user friendly.

3.9.0 **SYSTEM DESIGN SPECIFICATION**

The Model Driven Decision support system is modeled, developed, and implemented with due consideration to the rapidly changing trends in computing and information technology. It is essentially a window based application to enhance queue management and support resource allocation decisions.

3.9.1 **PROGRAMMING TOOLS**

The model –driven decision support system platform is built with the following tools;

- **C# .NET: C#** (pronounced *see sharp*) is a multi-paradigm programming language encompassing strong typing, imperative, declarative, functional, generic, object-oriented (class-based), and component-oriented programming disciplines. It was developed by Microsoft within its .NET Framework 4 initiative as part of the Visual Studio 2010 and later approved as a standard by Ecma (ECMA-334) and ISO (ISO/IEC 23270:2006). C# is one of the programming languages designed for the Common Language Infrastructure.

C# is intended to be a simple, modern, general-purpose, object-oriented programming language

- **Microsoft SQL Server**: MS SQL server is a relational database server, developed by Microsoft: It is a software product whose primary function is to store and retrieve data as requested by other software applications, be it those on the same computer or those running on another computer across a network (including the Internet).

3.10.0 **SYSTEM REQUIREMENTS SPECIFICATION**

The system requirements intend to define the specifications of the system to be designed. They involve a description of the system to be implemented, how it should run, application domain information, and constraints of the system's operation. The requirements specification is structured and formalized during analysis.

The requirements specification is one of the most important steps when designing the new system. These requirements provide more information regarding the system to make it possible to begin contemplating the conceptual model for the software engineering effort.

The principal objective of developing the model-Driven Decision Support System for Queue Management is to provide a decision support tool that will assist organizations in managing queuing situations and making optimal resource allocation decisions.

Therefore, to achieve this objective the system was designed to provide a simple and user-friendly graphical user interface that includes all functionalities of the queuing system. The decision support system was designed in such a way as to accept queue input parameters, perform the necessary analysis and produce output measures to assist the decision maker in making decisions about the queue situation. Based on the calculated outputs, the system will also make suggestions on the optimum number of service facilities suitable to manage the queue situation.

Due to the variety of hardware used in different organizations and sector that have queuing situations, the system would be designed to be compatible with different hosts (desktop, laptop, and notebook) on the client side and the enterprise server on the central side for various organization and systems.

Finally, the decision support system would be able to process data in real time and assure an acceptable level of usability based on the following hardware specifications.

3.10.1 **HARDWARE SPECIFICATION**

The minimum hardware requirements for effective performance of the proposed decision support system are:

• Computer CPU: Intel or compatible 400 MHz or higher;
• Memory (RAM): 256 MB;
• Hard Disk space: 60 GB or higher;
• Monitor: 800 X 600 or higher resolution;
• CD-ROM;
• Mouse: Microsoft or compatible.

3.10.2 **SOFTWARE REQUIREMENT**

For an optimum performance the decision support system requires a minimum of Microsoft Windows XP operating system and Microsoft SQL server Database Management Studio software.

CHAPTER FOUR

ANALYSIS AND SYSTEM DEVELOPMENT

4.0.0 **INTRODUCTION**

This chapter deals with the design which shall be further substantiated by actual development of the new system.

4.1.0 **SYSTEM DEVELOPMENT**

The development of a model- driven Decision support platform for queue management involved the following processes:

4.1.1 **Program Coding**

The coding of the program was done using modular approach to programming. The program codes are as shown on appendix A.

4.1.2 **Program Interface Design**

The interface design for this system was done to be suitable to the system's duties and to its users who will interact directly with the system. It incorporates the essential features of a decision support system like Authentication, system configuration, input screen, output interface, data management tools, user support etc.

The criteria used in designing the interface are as follows:

I. Easy to use: easy even with inexperienced users.

II. Easy to learn: easy for users to remember.

III. Fast processing and responding speed.

The interface designs are shown in the appendix B.

4.1.3 **Output Design**

The output was designed and presented in a way that is easy to understand and interpret. The output includes; server utilization factor, average number of customers in a queue, average

number customers in a system, average waiting time of customers in the system, average waiting time of customers in the system, and probability of an idle system etc.
The output designs are shown in appendix C.

4.2.0 **SYSTEM TESTING**

The various modules consisting the model-driven decision support system were run independently of each other (a system known as unit testing) before they are integrated as a single system or application. In this process all syntax, logical, run time and logical errors in each module were detected and debugged.

4.3.0 **SYSTEM INSTALLATION**

To install this new system, a computer system is required with at least a Microsoft windows XP operating system, and more preferably Windows 7 OS. The system would be reconfigured with the installation of Microsoft SQL server with the internet information service (IIS) of Microsoft windows operating system to be able to install and run this application over the internet or the World Wide Web.

4.4.0 **SYSTEM DOCUMENTATION**

Giving a written description of what the system is and how it can be used and modified is of great importance to users, operators, and programmers who may need to modify the program further.

USERS DOCUMENTATION

Each of the modules should be run one after the other to avoid overloading and eventual crashing of the software.

TECHNICAL/PROGRAMMERS DOCUMENTATION

The program or software has internal documentation to help other programmers. More so, modifying this software requires a good understanding of Queuing Theory, the .NET framework's C# programming language and Microsoft SQL database management application.

CHAPTER FIVE

SUMMARY, CONCLUSION AND RECOMMENDATION

5.0 SUMMARY OF THE STUDY

This study, Design and Implementation of a Model- Driven Decision Support Platform for Queue Management, is a study conceived to bring to bear how the new technological trend of Information Technology and decision support systems can be used to address the persistent problem of waiting lines (Queues) by translating mathematical models which abound in the research community and academia into useful, user – friendly applications to support management decision making functions. The study focused on the use of mathematical models from Queuing theory to design a decision support system software program to aid in decision making specifically, in the area of queue management across different sectors and fields of endeavor.

This project report discussed the design and implementation of a model- driven decision support platform for queue management that facilitates the specification, management, and analysis of queuing systems across diverse queuing systems and types. The system which implements models from queuing theory is composed of five subsystems which make up the decision support system that can be installed on a standalone system or a Client Server architecture enabling partial or total access to the database and providing real time data for decision making.

The application designed using the C# language is hosted on the Microsoft .NET framework 4.5 platform running on the Windows OS. The database was designed and supported on the Microsoft SQL server. The framework used in the design and modeling of the system is the Object-Oriented methodology which allows the use of principles such as inheritance, polymorphism and modularity. The system has been successfully tested and implemented on queuing systems in Healthcare, Banking and Retail marketing among others.

5.1.0 CONCLUSION

A study on the "Design of a Decision Support Platform for Queue Management" has been done in order to understand how the queuing system works in variety of human endeavor as well as to create the decision support system based on models from queuing theory. The further study on DSS model and architecture is done to ensure that Decision Support Platform for Queue Management follows the actual basic framework and model of a Decision Support Systems. The combination of Object Oriented Analysis (OOA) and design is applied in this model particularly in the database subsystem to generate better analysis of the queuing variables. The output from the system will be the result analysis and recommendation to the user. The analysis will be generated by the system's User Interface component where it will be presented to the users to aid them in their queue management decision making with the presentation of this fine grained information. However, it will only present the information and outputs derived from various queue theory models used in the system. The recommendation will judge, evaluate and provide recommendations based on the same outputs that are presented to the user in the analysis.

5.2.0 RECOMMENDATIONS

This study recommends the following suggestions that could foster effective management of queuing systems using decision support systems:

I. The battery of carefully and meticulously researched and developed mathematical models found in the academia specifically, in the fields of operations research and related sciences could be translated into user-friendly decision support software packages.

II. Maintenance operations on some software packages are expensive and not readily available. Microsoft platform oriented software are easy to use, easily maintained with less cost and readily available, and so recommended for development of decision support systems especially in developing countries like Nigeria.

5.3.0 SUGGESTIONS FOR FURTHER STUDY

This study can be improved upon by including automatically generated graphical presentation of analyzed results.

Moreover, design of model driven decision support systems should be given a system approach, as other operations can be integrated as one unified and complete decision application to incorporate cost related and optimality analysis.

REFERENCES

Abhijit A. P. and Ravindra K. A. (2007). Dynamic Ideas. Belmont, Massachusetts, ISBN: 0-9759146-4-2.

Alavi, M. and Joachimsthaler, E. A. (1992). Revisiting DSS Implementation Research: A Meta-Analysis of the literature and suggestions for researchers. *MIS Quarterly*, 16(1), 95-116.

Alter, S.L. (1975). *A Study of Computer Aided Decision Making in Organizations*. Ph.D. dissertation, M.I.T.

Alter, S.L. (1980). Decision Support Systems: Current Practice and Continuing Challenge. Reading, MA: Addison-Wesley.

Alter, S.L. (1977). Why Is Man-Computer Interaction Important for Decision Support Systems?. *Interfaces*, 7(2), 109-115.

Anon, (2009). Interagency Fuels Treatment Decision Support System Conceptual Design. 9-11.

Armstrong, M. P., Densham, P. J. and Rushton, G. (1986). Architecture for a microcomputer based spatial decision support system," Second International Symposium on Spatial Data Handling, 120(131): International Geographic Union.

Berners-Lee, T. (1996). The World Wide Web: Past, Present and Future.
URL http://www.w3.org/People/Berners-Lee/1996/ppf.html, last accessed March 5, 2007.

Bhargava, H. and Power, D. J. (2001). Decision Support Systems and Web Technologies: A Status Report. Proceedings of the 2001 Americas Conference on Information Systems, Boston, MA,

Blanchard, B. S. and Fabrycky, W. J. (1990). *Systems Engineering and Analysis*.Englewood Cliffs, N.J.: Prentice Hall.

Bonczek, R. H., Holsapple, C.W., and Whinston, A.B. (1981). Foundations of Decision Support Systems, New York: Academic Press.

Buchanan, B.G. and Shortliffe, E. H. (1984). Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project.

Bui, T. X. and Jarke, M. (1986). Communications Design for Co-op: A Group Decision Support System. *ACM Transactions on Office Information Systems*, 4(2), 81-103.

Codd, E.F., Codd, S.B. and Salley, C.T. (1993). Providing OLAP (On-Line Analytical Processing) to User-Analysts: An IT Mandate.

Crossland, M. D., Wynne, B. E. and Perkins, W. C. (1995). Spatial Decision Support Systems: An overview of technology and a test of efficacy. *Decision Support Systems*, 14(3), 219-235.

Davis, G. (1974). Management Information Systems: Conceptual Foundations, Structure, and Development. New York: McGraw-Hill, Inc.

DeSanctis, G. and Gallupe, R. B. (1987). A Foundation for the Study of Group Decision Support Systems. *Management Science*, 33(5), 589 - 609.

Devlin, B.A. (1997). Data Warehouse: From Architecture to Implementation. Addison-Wesley.

Devlin, B.A. and Murphy, P. T. (1988). An architecture for a business and information system. *IBM Systems Journal*, 27(1), 60-80.

Dhar, V. and Stein, R. (1997). Intelligent Decision Support Methods: The Science of Knowledge. Upper Saddle River, NJ: Prentice-Hall.

Donovan, J.J. and Madnick, S.E. (1977). Institutional and Ad Hoc DSS and Their Effective Use. *Data Base*, 8(3).

Fedorowicz, J. (1993). A Technology Infrastructure for Document-Based Decision Support Systems. Upper Saddle River, N.J: Prentice-Hall.

Ferguson, R. L. and Jones, C. H. (1969). A Computer Aided Decision System, *Management Science*, 15(10), 550-562.

Gerrity, T. P., Jr.(1971). Design of Man-Machine Decision Systems: An Application to Portfolio Management. *Sloan Management Review*, 12(2), 59-75.

Gray, P. (1983). Guide to IFPS (Interactive Financial Planning System), New York: McGraw-Hill Book Company.

Gray, P. (1981). The SMU decision room project, *Transactions of the Ist International Conference on Decision Support Systems* (Atlanta, Ga.), 122-129.

Hackathorn, R. D. and Keen, P. G. W. (1981). Organizational Strategies for Personal Computing in Decision Support Systems. *MIS Quarterly*, (5)3.

Holsapple, C.W. and Whinstone, A. B. (1996). Decision Support Systems: A Knowledge-Based Approach. St. Paul: West Publishing.

Houdeshel, G. and Watson, H. (1987). The Management Information and Decision Support (MIDS) System at Lockheed-Georgia. *MIS Quarterly*, 11(1), 127-140.

Keen, P. G. W. (1978). Decision support systems: an organizational perspective. Reading, Mass:Addison-Wesley Publishing.

Kersten, G.E. (1985). NEGO - Group Decision Support System, *Information and Management*, 8(5), 237-246.

Klein, M. and Methlie, L. B. (1995). Knowledge-based Decision Support Systems with Applications in Business. Chichester, UK: John Wiley and Sons.

Kossiakoff, A. and William N. S. (2003). Systems Engineering: Principles and Practices, 413.

Larson, R., (1987). Perspectives on Queues: Social Justice and the Psychology of Queuing. *Operations Research*, 3,895–905.

Little, J. D. C. (1970). Models and Managers: The Concept of a Decision Calculus. *Management Science*, 16(8), 466-485.

Little, J. D. C. (1975). Brandaid, an On-Line Marketing Mix Model, Part 2: Implementation, Calibration and Case Study. *Operations Research,* 23(4), 656-673.

Little, J.D.C. and Lodish, L.M. (1969). A Media Planning Calculus. *Operations Research,* 17, 1-35.

McCosh, A. M and ScottMorton, M. S.(1978). Management Decision Support Systems. London,uk: Macmillan.

McCosh, A. (2002). "Comments on 'A Brief History of DSS'," email to D. Power. http://dssresources.com/history/dsshistory.html , last accessed March 10, 2007.

McCosh, A. M. and Correa-Perez, B. A. (2006). Intelligent Decision-making Support Systems: Foundations, Applications and Challenges. *Springer-Verlag*, 475-494.

National Research Council. (1999). Funding a Revolution: Government Support for Computing Research.
http://www.nap.edu/readingroom/books/far/contents.html

Nylund, A. (1999). Tracing the BI Family Tree. *Knowledge Management*, 7(5), 23- 45.

Peter, B. (2006). A Survey of Object Oriented Methods," URL: http://www.freetutes.com/systemanalysis/sa2-object-oriented-methodology.html

Power, D. J. (2003). Decision Support Systems: Concepts and Resources for Managers, Westport, CT: Greenwood/Quorum.

Power, D. J. (2002). Decision Support Systems: Concepts and Resources for Managers, Westport, CT:Greenwood/Quorum.

Power, D. J. (2001). Supporting Decision-Makers: An Expanded Framework," In Harriger, A. (Editor), e-Proceedings Informing Science Conference, Krakow, Poland, 431-436.

Raymond, R.C. (1966). Use of the Time-sharing Computer in Business Planning and Budgeting. *Management Science*, 12(8), 363-381.

Redman, L.V. and Mory, A.V.H. (1923). The Romance of Research. *Encyclopedia of Social Sciences,* 8, 234-256.

Saaty, T. (1982). Decision Making for Leaders; the Analytical Hierarchy Process for Decisions in a Complex World. Wadsworth, Belmont, Calif.

ScottMorton, M. S. (1967). *Computer-Driven Visual Display Devices - Their Impact on the Management Decision-Making Process*. Doctoral Dissertation, Harvard Business School.

ScottMorton, M. S. and McCosh, A. M. (1968). Terminal Costing for Better Decisions. *Harvard Business Review,* 46(3), 147–56.

ScottMorton, M. S. (1966). Management Decision Systems; Computer-based support for decision making, Boston, Division of Research, Graduate School.

ScottMorton, M. S. and Stephens, J. A. (1968). The impact of interactive visual display systems on the management planning process. *IFIP Congress*, 2, 1178-1184.

ScottMorton, M. S. (1971). Management Decision Systems; Computer-based support for decision making,Boston, Division of Research, Graduate School of Business Administration, Harvard University.

Sharma, J. K. (2008). Operations Research Problems and Solutions. 3/Ed:Macmillan India Limited.

Sprague, R.H. and Carlson, E. D. (1982). Building effective decision support systems. Englewood Cliffs, N.J: Prentice-Hall. ISBN 0-130 – 86215 - 0.

Turoff, M. (1970). Delphi Conferencing: Computer Based Conferencing with Anonymity. *Journal of Technological Forecasting and Social Change*. 3(2), 159-204.

Wilson,B., Hurley, P. and Diffenderfer, P. (2011). Improving runway queue management: Modifying SDSS to accommodate deicing. *National Transportation System Center*, I6-1 - I6-12.

Young, P. V. (1966). Scientific Social Surveys and Research. Bombay, India: Prentice- Hall.

http://81.255.198.178/pdf/EN/ESII-WhitePaper_EN.pdf

http://en.wikipedia.org/wiki/Queue

http://www.m2ksys.com/documents/45.html

http://www.wavetec.com/queue-management-system.html
www.acftechnologies.com /wellington-fl-touts-efficient-service-with-new-kiosk/
http://www.informationbuilders.com/decision_support.html

http://searchcio.techtarget.com/definition/decision-support-system

http://www.microstrategy.com/decision-support-system.html

http://searchcio.techtarget.com/definition/decision-support-system

http://www.freetutes.com/systemanalysis/sa2-object-oriented-methodology.html

APPENDIX B: PROGRAM CODE LISTING

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace QueueManager
{
    public partial class frmMMSFinite : Form
    {
        public frmMMSFinite()
        {
            InitializeComponent();
        }

        private void queueVariableBindingNavigatorSaveItem_Click(object sender,
EventArgs e)
        {
            queueModelTextBox.Text = "MMSFinite";
            this.Validate();
            this.queueVariableBindingSource.EndEdit();
            this.tableAdapterManager.UpdateAll(this.queueManagerDataSet);

        }

        private void frmMMSFinite_Load(object sender, EventArgs e)
        {
            // TODO: This line of code loads data into the
'queueManagerDataSet.QueueVariable' table. You can move, or remove it, as needed.

this.queueVariableTableAdapter.FillByQueueModel(this.queueManagerDataSet.QueueVariabl
e, "MMSFinite");
```

```
}

private void btnSave_Click(object sender, EventArgs e)
{
    queueVariableBindingNavigatorSaveItem.PerformClick();
}

private double factorial(double number)
{
    double ans = 0;

    if (number > 0)
    {
        double count = 1;
        while (count <= number)
        {
            if (count == 1)
            {
                ans = 1;
                count++;
            }
            else
            {
                ans = count * ans;
                count++;
            }
        }
    }
    else if (number == 0)
    {
        ans = 1;
    }
    else if (number < 0)
    {
        MessageBox.Show("Please enter only a positive number.");
    }
    return ans;
}


private double P0(out double finalAnswer)
{
   double answer = 0;

    double arrivalRate = 0;
    double serviceRate = 0;
    double numberOfServers = 0;
    double numberOfCustomers = 0;

    double.TryParse(numberOfServersTextBox.Text, out numberOfServers);
    double.TryParse(arrivalRateTextBox.Text, out arrivalRate);
    double.TryParse(serviceRateTextBox.Text, out serviceRate);
    double.TryParse(queueCapacityTextBox.Text, out numberOfCustomers);
    double roh = Math.Round( arrivalRate / (numberOfServers * serviceRate));
```

```
            //double SummationFunction(out double summationValue)
            //{
            //    double summationAnswer = 0;
            //    summationValue = summationAnswer;
            //    return summationAnswer;
            //}

            //if roh = 1 then
            if (roh == 1)
            {
                double summation = 0;
                int count = 0;
                while (count <= numberOfServers - 1)
                {
                    summation = summation + (1 / factorial(count)) *
Math.Pow((numberOfServers * Math.Round (roh)), count);
                    count++;
                }
                double john = 0;
                john = (1 / factorial(numberOfServers)) *
(Math.Pow(Math.Round(arrivalRate / serviceRate), numberOfServers)) *
(numberOfCustomers - numberOfServers + 1);
                answer = summation + john;

                //answer = summation + (1 / factorial(numberOfServers) *
Math.Pow((arrivalRate / serviceRate), numberOfServers) * (numberOfServers *
serviceRate) / ((numberOfServers * serviceRate) - arrivalRate));

            }
            else if (Math.Round(roh) > 1)
            {

                double summationOne = 0;
                int countOne = 0;
               // while (countOne <= numberOfServers - 1)
                for( countOne = 0; countOne < (numberOfServers); countOne++)
                {
                    summationOne = summationOne + (1 / factorial(countOne)) *
Math.Pow(Math.Round(arrivalRate/serviceRate), countOne);
                    //countOne++;
                }

                double summationTwo = 0;
                int countTwo = Convert.ToInt32(numberOfServers);

                while (countTwo <= numberOfCustomers)
                {
                    summationTwo = summationTwo + (1/((factorial(numberOfServers) *
Math.Pow(numberOfServers, (countTwo - numberOfServers)))) * Math.Pow(Math.Round(
arrivalRate/serviceRate), countTwo));
                    countTwo++;
                }

                answer = summationOne + summationTwo;
```

```
            }


            finalAnswer = Math.Pow(answer, -1);
            return finalAnswer;
        }

        private double P1(out double result)
        {

            double arrivalRate = 0;
            double serviceRate = 0;
            double numberOfServers = 0;
            double numberOfCustomers = 0;

            double.TryParse(numberOfServersTextBox.Text, out numberOfServers);
            double.TryParse(arrivalRateTextBox.Text, out arrivalRate);
            double.TryParse(serviceRateTextBox.Text, out serviceRate);
            double.TryParse(queueCapacityTextBox.Text, out numberOfCustomers);
            double roh = arrivalRate / (numberOfServers * serviceRate);

            double summationThree = 0;
            int countThree = 0;
            while (countThree <= numberOfServers - 1)
            {
                summationThree = summationThree + ((numberOfServers - countThree) /
factorial(countThree)) * Math.Pow(Math.Round(arrivalRate / serviceRate), countThree);
                countThree++;
            }

            result =  summationThree;

            return result;

        }


        private void btnAnalyse_Click(object sender, EventArgs e)
        {
            double arrivalRate = 0;
            double serviceRate = 0;
            double numberOfServers = 0;
            double QueueCapacity = 0;

            double.TryParse(numberOfServersTextBox.Text, out numberOfServers);
            double.TryParse(arrivalRateTextBox.Text, out arrivalRate);
            double.TryParse(serviceRateTextBox.Text, out serviceRate);
            double.TryParse(queueCapacityTextBox.Text, out QueueCapacity);

            //Server Utilization
            double serverUtilisation = Math.Round(arrivalRate / serviceRate);
            decimal serverUtill = Convert.ToDecimal(serverUtilisation);
            serverUtilisationTextBox.Text = decimal.Round(serverUtill, 2).ToString();

            //Average Number Of Customers in the Queue
```

```
            double calculatedfinalAnswer = 0;

            //double sRoh = 0;
            // double rohNSPlus = 0;
            // double salem = 0;
            //double rohNS = 0;


            // sRoh = ((Math.Pow((numberOfServers * serverUtilisation),
numberOfServers)) * serverUtilisation)/(factorial(numberOfServers) * Math.Pow((1-
serverUtilisation), 2));
            //rohNSPlus = (1 - (Math.Pow(serverUtilisation, (QueueCapacity -
numberOfServers + 1))));
            // salem =(1 - serverUtilisation) * (QueueCapacity - numberOfServers +
1);
            // rohNS = Math.Pow(serverUtilisation, (QueueCapacity -
numberOfServers));

            double aveQueueCustomers = 0;
            double roh = (arrivalRate / (numberOfServers * serviceRate));
            double count = numberOfServers;
            while (count <= QueueCapacity)
            {
                aveQueueCustomers = (count - numberOfServers) *
((Math.Pow((numberOfServers * roh), count)) / (factorial(numberOfServers) *
Math.Pow(numberOfServers, (count - numberOfServers)))) * P0(out
calculatedfinalAnswer);

                count++;

            }

                        decimal aveQueCust = Convert.ToDecimal(aveQueueCustomers);
            averageCustomersInQueueTextBox.Text = decimal.Round(aveQueCust,
2).ToString();
        }

        //Analyse
        private void Analysis(double arrivalRate, double serviceRate, double
numberOfServers, double QueueCapacity)

        {
            arrivalRate = 0;
            serviceRate = 0;
            numberOfServers = 0;
            QueueCapacity = 0;

            double.TryParse(txtNumberOfServers.Text, out numberOfServers);
            double.TryParse(txtArrivalRate.Text, out arrivalRate);
            double.TryParse(txtServiceRate.Text, out serviceRate);
            double.TryParse(txtQueueCapacity.Text, out QueueCapacity);

            //Server Utilization
            double serverUtilisation = Math.Round(arrivalRate / serviceRate);
            decimal serverUtill = Convert.ToDecimal(serverUtilisation);
```

```
            txtServerUtilisation.Text  = decimal.Round(serverUtill, 2).ToString();


            //Average Number Of Customers in the Queue
            double calculatedfinalAnswer = 0;

            //double sRoh = 0;
            // double rohNSPlus = 0;
            // double salem = 0;
            //double rohNS = 0;


            // sRoh = ((Math.Pow((numberOfServers * serverUtilisation),
numberOfServers)) * serverUtilisation)/(factorial(numberOfServers) * Math.Pow((1-
serverUtilisation), 2));
            //rohNSPlus = (1 - (Math.Pow(serverUtilisation, (QueueCapacity -
numberOfServers + 1))));
            // salem =(1 - serverUtilisation) * (QueueCapacity - numberOfServers +
1);
            // rohNS = Math.Pow(serverUtilisation, (QueueCapacity -
numberOfServers));

            double aveQueueCustomers = 0;
            double roh = (arrivalRate / (numberOfServers * serviceRate));
            double count = numberOfServers;
            while (count <= QueueCapacity)
            {
                aveQueueCustomers = (count - numberOfServers) *
((Math.Pow((numberOfServers * roh), count)) / (factorial(numberOfServers) *
Math.Pow(numberOfServers, (count - numberOfServers)))) * P0(out
calculatedfinalAnswer);

                count++;
            }


              decimal aveQueCust = Convert.ToDecimal(aveQueueCustomers);
             txtAverageCustomersInAQueue.Text = decimal.Round(aveQueCust,
3).ToString();


        }

        private void txtArrivalRate_TextChanged(object sender, EventArgs e)
        {
            if (!string.IsNullOrEmpty(txtArrivalRate.Text) &&
!string.IsNullOrEmpty(txtServiceRate.Text) &&
!string.IsNullOrEmpty(txtQueueCapacity.Text) &&
!string.IsNullOrEmpty(txtNumberOfServers.Text))
            {
                double arrivalRate2 = Convert.ToDouble(txtArrivalRate.Text);
                double serviceRate2 = Convert.ToDouble(txtServiceRate.Text);
                double numberOfServers2 = Convert.ToDouble(txtNumberOfServers.Text);
                double queueCapacity2 = Convert.ToDouble(txtQueueCapacity.Text);
```

```
                Analysis(arrivalRate2, serviceRate2, numberOfServers2,
queueCapacity2);
            }
        }

        private void txtServiceRate_TextChanged(object sender, EventArgs e)
        {
            if (!string.IsNullOrEmpty(txtArrivalRate.Text) &&
!string.IsNullOrEmpty(txtServiceRate.Text) &&
!string.IsNullOrEmpty(txtQueueCapacity.Text) &&
!string.IsNullOrEmpty(txtNumberOfServers.Text))
            {
                double arrivalRate2 = Convert.ToDouble(txtArrivalRate.Text);
                double serviceRate2 = Convert.ToDouble(txtServiceRate.Text);
                double numberOfServers2 = Convert.ToDouble(txtNumberOfServers.Text);
                double queueCapacity2 = Convert.ToDouble(txtQueueCapacity.Text);
                Analysis(arrivalRate2, serviceRate2, numberOfServers2,
queueCapacity2);
            }
        }

        private void txtQueueCapacity_TextChanged(object sender, EventArgs e)
        {
            if (!string.IsNullOrEmpty(txtArrivalRate.Text) &&
!string.IsNullOrEmpty(txtServiceRate.Text) &&
!string.IsNullOrEmpty(txtQueueCapacity.Text) &&
!string.IsNullOrEmpty(txtNumberOfServers.Text))
            {
                double arrivalRate2 = Convert.ToDouble(txtArrivalRate.Text);
                double serviceRate2 = Convert.ToDouble(txtServiceRate.Text);
                double numberOfServers2 = Convert.ToDouble(txtNumberOfServers.Text);
                double queueCapacity2 = Convert.ToDouble(txtQueueCapacity.Text);
                Analysis(arrivalRate2, serviceRate2, numberOfServers2,
queueCapacity2);
            }
        }

        private void txtNumberOfServers_TextChanged(object sender, EventArgs e)
        {
            if (!string.IsNullOrEmpty(txtArrivalRate.Text) &&
!string.IsNullOrEmpty(txtServiceRate.Text) &&
!string.IsNullOrEmpty(txtQueueCapacity.Text) &&
!string.IsNullOrEmpty(txtNumberOfServers.Text))
            {
                double arrivalRate2 = Convert.ToDouble(txtArrivalRate.Text);
                double serviceRate2 = Convert.ToDouble(txtServiceRate.Text);
                double numberOfServers2 = Convert.ToDouble(txtNumberOfServers.Text);
                double queueCapacity2 = Convert.ToDouble(txtQueueCapacity.Text);
                Analysis(arrivalRate2, serviceRate2, numberOfServers2,
queueCapacity2);
            }
        }
    }

}
```

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace QueueManager
{
    public partial class frmMM1Finite : Form
    {
        public frmMM1Finite()
        {
            InitializeComponent();
        }

        private void queueVariableBindingNavigatorSaveItem_Click(object sender,
EventArgs e)
        {
            queueModelTextBox.Text = "MM1Finite";
            this.Validate();
            this.queueVariableBindingSource.EndEdit();
            this.tableAdapterManager.UpdateAll(this.queueManagerDataSet);

        }

        private void frmMM1Finite_Load(object sender, EventArgs e)
        {
            // TODO: This line of code loads data into the
'queueManagerDataSet.QueueVariable' table. You can move, or remove it, as needed.
          //
this.queueVariableTableAdapter.Fill(this.queueManagerDataSet.QueueVariable);
            // TODO: This line of code loads data into the
'queueManagerDataSet.QueueVariable' table. You can move, or remove it, as needed.

this.queueVariableTableAdapter.FillByQueueModel(this.queueManagerDataSet.QueueVariabl
e, "MM1Finite");

        }

        private double PN(out double answer)
        {
            answer = 0;
            double serverUtil = Convert.ToDouble(arrivalRateTextBox.Text) /
Convert.ToDouble(serviceRateTextBox.Text);
            answer = (1 - serverUtil) / (1 - Math.Pow(serverUtil,
Convert.ToDouble(queueCapacityTextBox.Text) + 1)) * Math.Pow(serverUtil,
Convert.ToDouble(queueCapacityTextBox.Text));
            return answer;
        }
```

```
        private double AnalysisPN(out double answer)
        {
            answer = 0;
            double serverUtil = Convert.ToDouble(txtArrivalRate.Text) /
Convert.ToDouble(txtServiceRate.Text);
            answer = (1 - serverUtil) / (1 - Math.Pow(serverUtil,
Convert.ToDouble(txtQueueCapacity.Text) + 1)) * Math.Pow(serverUtil,
Convert.ToDouble(txtQueueCapacity.Text));
            return answer;
        }

        private void btnAnalyse_Click(object sender, EventArgs e)
        {
            double arrivalRate = 0;
            double serviceRate = 0;
            double queueCapacity = 0;

            double.TryParse(queueCapacityTextBox.Text, out queueCapacity);
            double.TryParse(arrivalRateTextBox.Text, out arrivalRate);
            double.TryParse(serviceRateTextBox.Text, out serviceRate);

            //Server Utilization
            double serverUtilisation = arrivalRate / serviceRate;
            decimal serverUtill = Convert.ToDecimal(serverUtilisation);
            serverUtilisationTextBox.Text = decimal.Round(serverUtill, 2).ToString();
            double aveSystemCustomers = 0;
            if (serverUtill == 1)
            {
                //Average Number of Customers in the system
                aveSystemCustomers = queueCapacity / 2;

            }
            else
            {
                //Average Number of Customers in the system
                aveSystemCustomers = (serverUtilisation / (1 - serverUtilisation)) -
((queueCapacity + 1) * Math.Pow(serverUtilisation, (queueCapacity + 1))) / (1 -
Math.Pow(serverUtilisation, (queueCapacity + 1)));
            }

            decimal aveSysCustomers = Convert.ToDecimal(aveSystemCustomers);
            averageCustomersInSystemTextBox.Text = decimal.Round(aveSysCustomers,
0).ToString();


            //Average Number Of Customers in the Queue
            double aveQueueCustomers = Convert.ToDouble(aveSysCustomers) -
(arrivalRate / serviceRate);
            decimal aveQueCust = Convert.ToDecimal(aveQueueCustomers);
            averageCustomersInQueueTextBox.Text = decimal.Round(aveQueCust,
0).ToString();

            //Average waiting time of customer in the system
            double calculatedPN = 0;
```

```
            double averageWaitingTimeInSystem = Convert.ToDouble(aveSysCustomers) /
arrivalRate * (1.0 - PN(out calculatedPN));

            decimal aveSystemWaiting = Convert.ToDecimal(averageWaitingTimeInSystem);
            averageWaitingTimeInSystemTextBox.Text = decimal.Round(aveSystemWaiting,
2).ToString();


            //Aeverage waiting time of customers in queue
            double averageWaitingTimeQueue = Convert.ToDouble(aveSystemWaiting) - (1
/ serviceRate);
            decimal aveQueueWaiting = Convert.ToDecimal(averageWaitingTimeQueue);
            averageWaitingTimeInQueueTextBox.Text = decimal.Round(aveQueueWaiting,
2).ToString();


            //Fluctuation of queue length
            double fluctuationOfQueueLength = (arrivalRate * serviceRate) /
Math.Pow((serviceRate - arrivalRate), 2);
            decimal queueFluctuation = Convert.ToDecimal(fluctuationOfQueueLength);
            queueFluctuationTextBox.Text = decimal.Round(queueFluctuation,
2).ToString();


            //Probability Of Non Empty Queue
            double probabilityOfNonEmptyQueue = Math.Pow((arrivalRate / serviceRate),
2);
            decimal nonEmptyQueueProbability =
Convert.ToDecimal(probabilityOfNonEmptyQueue);
            probabilityOfNonEmptyQueueTextBox.Text =
decimal.Round(nonEmptyQueueProbability, 2).ToString();
        }

        private void btnSave_Click(object sender, EventArgs e)
        {
            queueVariableBindingNavigatorSaveItem.PerformClick();
        }



        //Analysis

        private void Analysis(double arrivalRate, double serviceRate, double
queueCapacity)
        {
             arrivalRate = 0;
            serviceRate = 0;
            queueCapacity = 0;

            double.TryParse(txtQueueCapacity.Text, out queueCapacity);
            double.TryParse(txtArrivalRate.Text, out arrivalRate);
            double.TryParse(txtServiceRate.Text, out serviceRate);

            //Server Utilization
            double serverUtilisation = arrivalRate / serviceRate;
```

```
            decimal serverUtill = Convert.ToDecimal(serverUtilisation);
            txtServerUtilisation.Text = decimal.Round(serverUtill, 2).ToString();
            double aveSystemCustomers = 0;
            if (serverUtill == 1)
            {
                //Average Number of Customers in the system
                aveSystemCustomers = queueCapacity / 2;

            }
            else
            {
                //Average Number of Customers in the system
                aveSystemCustomers = (serverUtilisation / (1 - serverUtilisation)) -
((queueCapacity + 1) * Math.Pow(serverUtilisation, (queueCapacity + 1))) / (1 -
Math.Pow(serverUtilisation, (queueCapacity + 1)));
            }

            decimal aveSysCustomers = Convert.ToDecimal(aveSystemCustomers);
            txtAverageCustomersInSystem.Text = decimal.Round(aveSysCustomers,
0).ToString();


            //Average Number Of Customers in the Queue
            double aveQueueCustomers = Convert.ToDouble(aveSysCustomers) -
(arrivalRate / serviceRate);
            decimal aveQueCust = Convert.ToDecimal(aveQueueCustomers);
            txtAverageCustomersInAQueue.Text = decimal.Round(aveQueCust,
0).ToString();

            //Average waiting time of customer in the system
            double calculatedPN = 0;
            double averageWaitingTimeInSystem = Convert.ToDouble(aveSysCustomers) /
arrivalRate * (1.0 - AnalysisPN(out calculatedPN));

            decimal aveSystemWaiting = Convert.ToDecimal(averageWaitingTimeInSystem);
            txtAverageWaitingTimeInSystem.Text = decimal.Round(aveSystemWaiting,
2).ToString();

            //Aeverage waiting time of customers in queue
            double averageWaitingTimeQueue = Convert.ToDouble(aveSystemWaiting) - (1
/ serviceRate);
            decimal aveQueueWaiting = Convert.ToDecimal(averageWaitingTimeQueue);
            txtAverageWaitingTimeInQueue.Text= decimal.Round(aveQueueWaiting,
2).ToString();

            //Fluctuation of queue length
            double fluctuationOfQueueLength = (arrivalRate * serviceRate) /
Math.Pow((serviceRate - arrivalRate), 2);
            decimal queueFluctuation = Convert.ToDecimal(fluctuationOfQueueLength);
            txtQueueFluctuation.Text = decimal.Round(queueFluctuation, 2).ToString();


            //Probability Of Non Empty Queue
            double probabilityOfNonEmptyQueue = Math.Pow((arrivalRate / serviceRate),
2);
```

```
            decimal nonEmptyQueueProbability =
Convert.ToDecimal(probabilityOfNonEmptyQueue);
            txtProbabilityOfNonEmptyQueue.Text =
decimal.Round(nonEmptyQueueProbability, 2).ToString();
        }

        private void averageWaitingTimeInSystemTextBox_TextChanged(object sender,
EventArgs e)
        {

        }

        private void arrivalRateTextBox_TextChanged(object sender, EventArgs e)
        {
            //if (!string.IsNullOrEmpty(arrivalRateTextBox.Text))
            //{
            //    btnAnalyse.PerformClick();
            //}
        }

        private void txtArrivalRate_TextChanged(object sender, EventArgs e)
        {
            if (!string.IsNullOrEmpty(txtArrivalRate.Text) &&
!string.IsNullOrEmpty(txtServiceRate.Text) &&
!string.IsNullOrEmpty(txtQueueCapacity.Text))
            {
                double arrivalRate2 = Convert.ToDouble(txtArrivalRate.Text);
                double serviceRate2 = Convert.ToDouble(txtServiceRate.Text);
                double queueCapacity2 = Convert.ToDouble(queueCapacityTextBox.Text);

                Analysis(arrivalRate2, serviceRate2, queueCapacity2);
                //btnAnalyse.PerformClick();
            }
        }

        private void txtServiceRate_TextChanged(object sender, EventArgs e)
        {
            if (!string.IsNullOrEmpty(txtArrivalRate.Text) &&
!string.IsNullOrEmpty(txtServiceRate.Text) &&
!string.IsNullOrEmpty(txtQueueCapacity.Text))
            {
                double arrivalRate2 = Convert.ToDouble(txtArrivalRate.Text);
                double serviceRate2 = Convert.ToDouble(txtServiceRate.Text);
                double queueCapacity2 = Convert.ToDouble(queueCapacityTextBox.Text);

                Analysis(arrivalRate2, serviceRate2, queueCapacity2);
               // btnAnalyse.PerformClick();
            }
        }

        private void txtQueueCapacity_TextChanged(object sender, EventArgs e)
        {
            if (!string.IsNullOrEmpty(txtArrivalRate.Text) &&
!string.IsNullOrEmpty(txtServiceRate.Text) &&
!string.IsNullOrEmpty(txtQueueCapacity.Text))
```

```
            {
                double arrivalRate2 = Convert.ToDouble(txtArrivalRate.Text);
                double serviceRate2 = Convert.ToDouble(txtServiceRate.Text);
                double queueCapacity2 = Convert.ToDouble(queueCapacityTextBox.Text);

                Analysis(arrivalRate2, serviceRate2, queueCapacity2);

               // btnAnalyse.PerformClick();
            }
        }

        private void button1_Click(object sender, EventArgs e)
        {

        }

        private void groupBox7_Enter(object sender, EventArgs e)
        {

        }

        //private void txtArrivalRate_TextChanged(object sender, EventArgs e)
        //{

        //}

        //private void AnalysisFunction()
        //{
        //    txtArrivalRate.Text = arrivalRateTextBox.Text;
        //    txtServiceRate.Text = serviceRateTextBox.Text;
        //    txtNumberOfServers.Text = numberOfServersTextBox.Text;
        //    txtQueueCapacity.Text = queueCapacityTextBox.Text;
        //}

        //private void textBox8_TextChanged(object sender, EventArgs e)
        //{

        //}

         }
}
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace QueueManager
{
    public partial class frmMM1FinitePopulation : Form
    {
        public frmMM1FinitePopulation()
```

```
        {
            InitializeComponent();
        }

        private void queueVariableBindingNavigatorSaveItem_Click(object sender,
EventArgs e)
        {
            queueModelTextBox.Text = "MM1FinitePopulation";
            this.Validate();
            this.queueVariableBindingSource.EndEdit();
            this.tableAdapterManager.UpdateAll(this.queueManagerDataSet);

        }

        private void frmMM1FinitePopulation_Load(object sender, EventArgs e)
        {
            // TODO: This line of code loads data into the
'queueManagerDataSet.QueueVariable' table. You can move, or remove it, as needed.

this.queueVariableTableAdapter.FillByQueueModel(this.queueManagerDataSet.QueueVariabl
e, "MM1FinitePopulation");

        }

        private double factorial(double number)
        {
            double ans = 0;

            if (number > 0)
            {
                double count = 1;
                while (count <= number)
                {
                    if (count == 1)
                    {
                        ans = 1;
                        count++;
                    }
                    else
                    {
                        ans = count * ans;
                        count++;
                    }
                }
            }
            else if (number == 0)
            {
                ans = 1;
            }
            else if (number < 0)
            {
                MessageBox.Show("Please enter only a positive number.");
            }
            return ans;
        }
```

```
        private double P0(out double finalAnswer)
        {
            double answer = 0;

            double arrivalRate = 0;
            double serviceRate = 0;
            double numberOfServers = 0;
            double numberOfCustomers = 0;
            double callingPopulation = 0;

            double.TryParse(numberOfServersTextBox.Text, out numberOfServers);
            double.TryParse(callingPopulationTextBox.Text, out callingPopulation);
            double.TryParse(arrivalRateTextBox.Text, out arrivalRate);
            double.TryParse(serviceRateTextBox.Text, out serviceRate);
            double.TryParse(queueCapacityTextBox.Text, out numberOfCustomers);
            double roh = arrivalRate / (numberOfServers * serviceRate);

            double summation = 0;
            int count = 0;
            while (count <= callingPopulation)
            {
                summation = summation + (factorial(callingPopulation) /
(factorial(callingPopulation - count))) * (Math.Pow((arrivalRate / serviceRate),
count));
                count++;
            }

            answer = summation;
            finalAnswer = Math.Pow(answer, -1);

            return finalAnswer;
        }


        private void btnAnalyse_Click(object sender, EventArgs e)
        {
            double arrivalRate = 0;
            double serviceRate = 0;
            double numberOfServers = 0;
            double numberOfCustomers = 0;
            double callingPopulation = 0;


            double.TryParse(numberOfServersTextBox.Text, out numberOfServers);
            double.TryParse(callingPopulationTextBox.Text, out callingPopulation);
            double.TryParse(arrivalRateTextBox.Text, out arrivalRate);
            double.TryParse(serviceRateTextBox.Text, out serviceRate);
            double.TryParse(queueCapacityTextBox.Text, out numberOfCustomers);
            double roh = arrivalRate / (numberOfServers * serviceRate);

            // Server utilisation
            double serverUtilisation = 0;
            serverUtilisation = arrivalRate / serviceRate;
            decimal serverUtill = Convert.ToDecimal(serverUtilisation);
```

```
            serverUtilisationTextBox.Text = decimal.Round(serverUtill, 2).ToString();

            // Probability of Non-Empty Queue

            double probabilityOfNonEmptyQueue = 0;
            probabilityOfNonEmptyQueue = Math.Pow((arrivalRate / serviceRate), 2);
            decimal nonEmptyProbability =
Convert.ToDecimal(probabilityOfNonEmptyQueue);
            probabilityOfNonEmptyQueueTextBox.Text =
decimal.Round(nonEmptyProbability, 2).ToString();


            // Fluctuation Of Queue Lenght( Queue Variance)
            double queueFluctuation = 0;
            queueFluctuation = (arrivalRate * serviceRate) / Math.Pow((serviceRate -
arrivalRate), 2);
            decimal queueVariance = Convert.ToDecimal(queueFluctuation);
            queueFluctuationTextBox.Text = decimal.Round(queueVariance,
2).ToString();


            // Average number of Customers in Queue

            double customersInQueue = 0;
            customersInQueue = callingPopulation - ((arrivalRate + serviceRate) /
arrivalRate) * (1 - P0(out customersInQueue));
            decimal aveQueuecust = Convert.ToDecimal(customersInQueue);
            averageCustomersInQueueTextBox.Text = decimal.Round(aveQueuecust,
2).ToString();


            //Average Number of Customers in System
            double customerInSystem = 0;
            customerInSystem = callingPopulation - ((serviceRate / arrivalRate) * (1
- P0(out customerInSystem)));
            decimal aveSystemcust = Convert.ToDecimal(customerInSystem);
            averageCustomersInSystemTextBox.Text = decimal.Round(aveSystemcust,
2).ToString();

            // Average Waiting Time in the Quueue
            double waitingTimeInQueue = 0;
            waitingTimeInQueue = customersInQueue / (arrivalRate * (callingPopulation
- customerInSystem));
            decimal aveWaitingTimeInQueue = Convert.ToDecimal(waitingTimeInQueue);
            averageWaitingTimeInQueueTextBox.Text =
decimal.Round(aveWaitingTimeInQueue, 2).ToString();

            // Avarage Waiting Time in The System
            double waitingTimeInSystem = 0;
            waitingTimeInSystem = waitingTimeInQueue + (1 / serviceRate);
            decimal aveWaitingTimeInSystem = Convert.ToDecimal(waitingTimeInSystem);
            averageWaitingTimeInSystemTextBox.Text =
decimal.Round(aveWaitingTimeInSystem, 2).ToString();
```

```
        }

        private void btnSave_Click(object sender, EventArgs e)
        {
            queueVariableBindingNavigatorSaveItem.PerformClick();
        }
    }
}

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace QueueManager
{
    public partial class frmMultiPhaseErlang : Form
    {
        public frmMultiPhaseErlang()
        {
            InitializeComponent();
        }

        private void queueVariableBindingNavigatorSaveItem_Click(object sender,
EventArgs e)
        {
            queueModelTextBox.Text = "MultiPhaseErlang";
            this.Validate();
            this.queueVariableBindingSource.EndEdit();
            this.tableAdapterManager.UpdateAll(this.queueManagerDataSet);

        }

        private void frmMultiPhaseErlang_Load(object sender, EventArgs e)
        {
            // TODO: This line of code loads data into the
'queueManagerDataSet.QueueVariable' table. You can move, or remove it, as needed.

this.queueVariableTableAdapter.FillByQueueModel(this.queueManagerDataSet.QueueVariabl
e, "MultiPhaseErlang");

        }

        private void btnAnalyse_Click(object sender, EventArgs e)
        {
            double arrivalRate = 0;
            double serviceRate = 0;
            double numberOfServers = 0;
            double numberOfCustomers = 0;

            double.TryParse(numberOfServersTextBox.Text, out numberOfServers);
```

```
            double.TryParse(arrivalRateTextBox.Text, out arrivalRate);
            double.TryParse(serviceRateTextBox.Text, out serviceRate);
            double.TryParse(queueCapacityTextBox.Text, out numberOfCustomers);
            double roh = arrivalRate / (numberOfServers * serviceRate);

            // Expected Number of Phases in the Queue
            double NumberOfPhasesInQueue = 0;
            NumberOfPhasesInQueue = ((numberOfServers + 1) / 2) * (arrivalRate /
(serviceRate * (serviceRate - arrivalRate)));
            decimal NumOfQueuePhases = Convert.ToDecimal(NumberOfPhasesInQueue);
            averagePhasesInQueueTextBox.Text = decimal.Round(NumOfQueuePhases,
3).ToString();

            //// Non Empty Queue Textbox= Expected Number of Phases in the system
            //// Number of Servers textbox = Number of phases


            //Expected Number of phases (Not Custumers) in the System
            double averageNumberOfPhasesInSystem = 0;
            averageNumberOfPhasesInSystem = ((numberOfServers + 1) / 2) *
(arrivalRate / (serviceRate - arrivalRate));
            decimal numberOfPhases =
Convert.ToDecimal(averageNumberOfPhasesInSystem);
            probabilityOfNonEmptyQueueTextBox.Text = decimal.Round(numberOfPhases,
3).ToString();


            // Expected Number of Customers in the Queue (not Phases)

            double averageCustomersInQueue = 0;
            averageCustomersInQueue = ((numberOfServers + 1) / (2 * numberOfServers))
* (Math.Pow(arrivalRate, 2) / (serviceRate * (serviceRate - arrivalRate)));
            decimal aveCustInQueue = Convert.ToDecimal(averageCustomersInQueue);
            averageCustomersInQueueTextBox.Text = decimal.Round(aveCustInQueue,
3).ToString();

            // Expected Number of customers In the System
            double averageCustomersInSystem = 0;
            averageCustomersInSystem = averageCustomersInQueue + (arrivalRate /
serviceRate);
            decimal aveCustInSystem = Convert.ToDecimal(averageCustomersInSystem);
            averageCustomersInSystemTextBox.Text = decimal.Round(aveCustInSystem,
3).ToString();

            // Average Waiting Time in Queue
            double averageWaitingTimeInQueue = 0;
            averageWaitingTimeInQueue = averageCustomersInQueue / arrivalRate;
            decimal aveWaitingQueue = Convert.ToDecimal(averageWaitingTimeInQueue);
            averageWaitingTimeInQueueTextBox.Text = decimal.Round(aveWaitingQueue,
3).ToString();

            // Average Waiting Time In system
            double averageWaitingTimeInSystem = 0;
            averageWaitingTimeInSystem = averageWaitingTimeInQueue + (1 /
serviceRate);
```

```
            decimal aveWaitingInSystem =
Convert.ToDecimal(averageWaitingTimeInSystem);
            averageWaitingTimeInSystemTextBox.Text =
decimal.Round(aveWaitingInSystem, 3).ToString();

            //Fluctuation of queue length
            double fluctuationOfQueueLength = (arrivalRate * serviceRate) /
Math.Pow((serviceRate - arrivalRate), 2);
            decimal queueFluctuation = Convert.ToDecimal(fluctuationOfQueueLength);
            queueFluctuationTextBox.Text = decimal.Round(queueFluctuation,
3).ToString();
            //Server Utilization
            double serverUtilisation = arrivalRate / serviceRate;
            decimal serverUtill = Convert.ToDecimal(serverUtilisation);
            serverUtilisationTextBox.Text = decimal.Round(serverUtill, 2).ToString();
        }

        private void btnSave_Click(object sender, EventArgs e)
        {
            queueVariableBindingNavigatorSaveItem.PerformClick();
        }
    }
}
```

APPENDIX C: DECISION SUPPORT SYSTEM SCREEN SHOTS

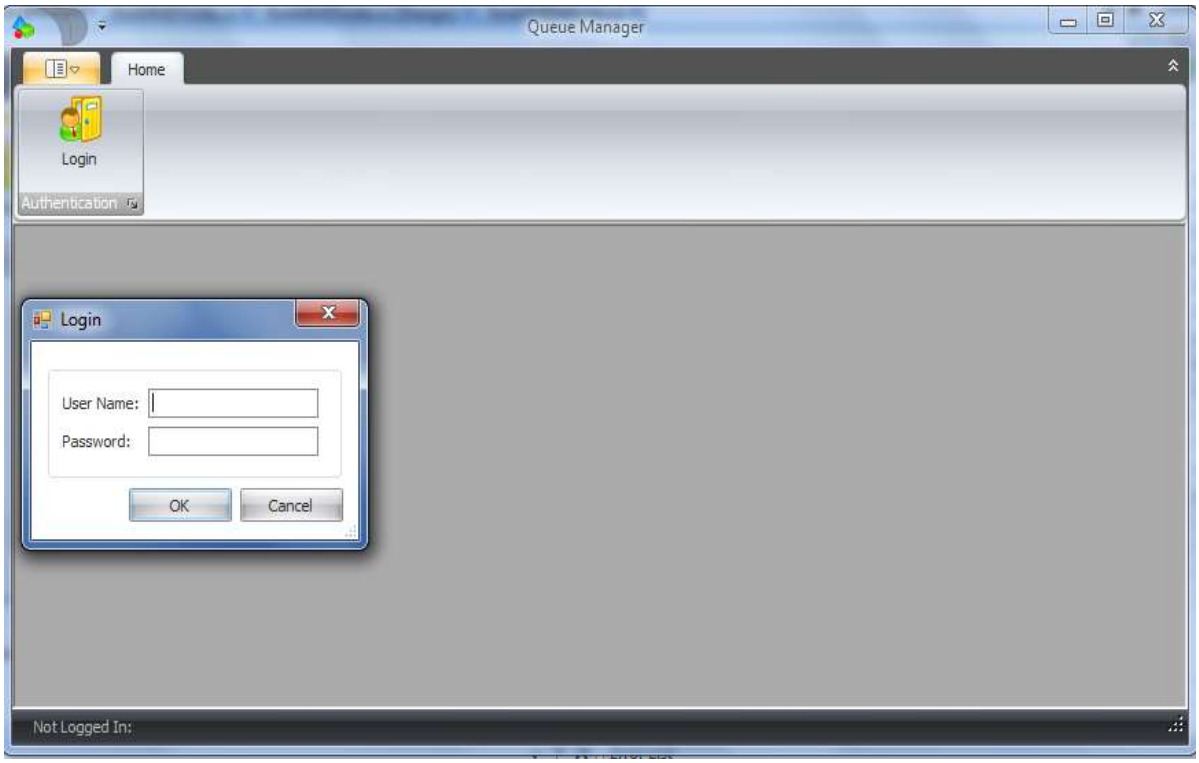


Fig.1: The Login Module Interface

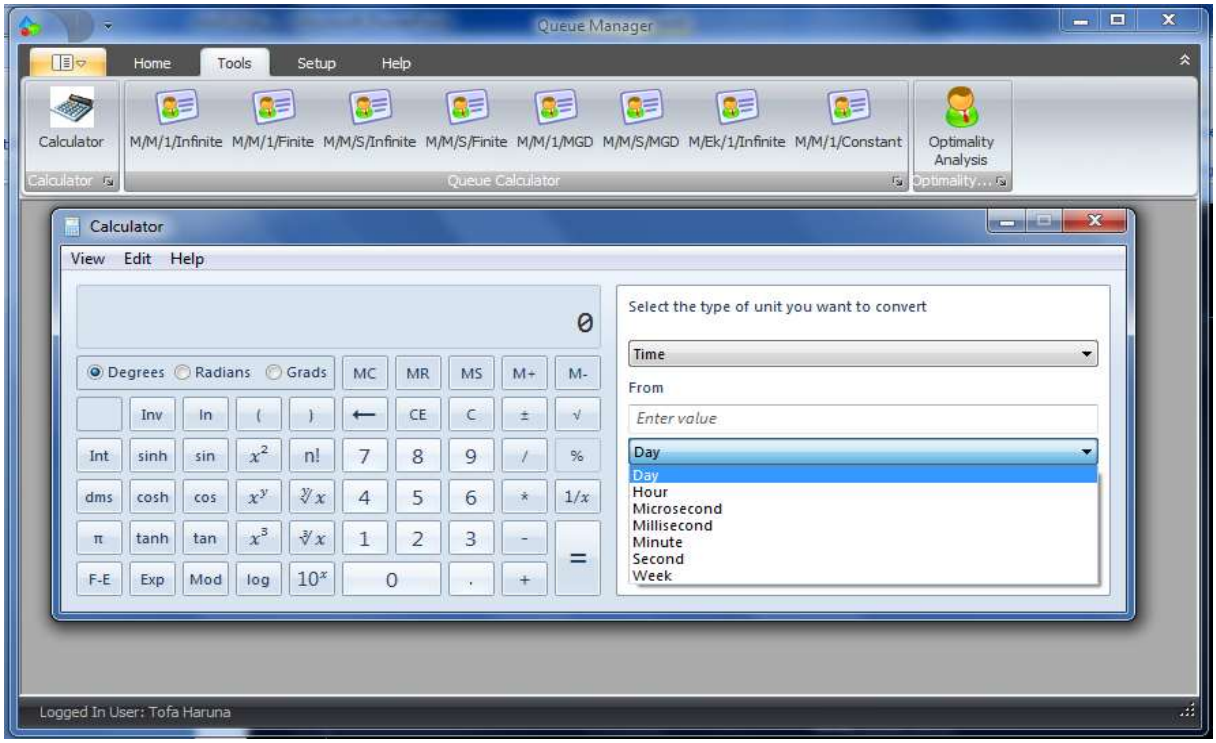


Fig. 2: The Main Screen Showing the Calculator tool

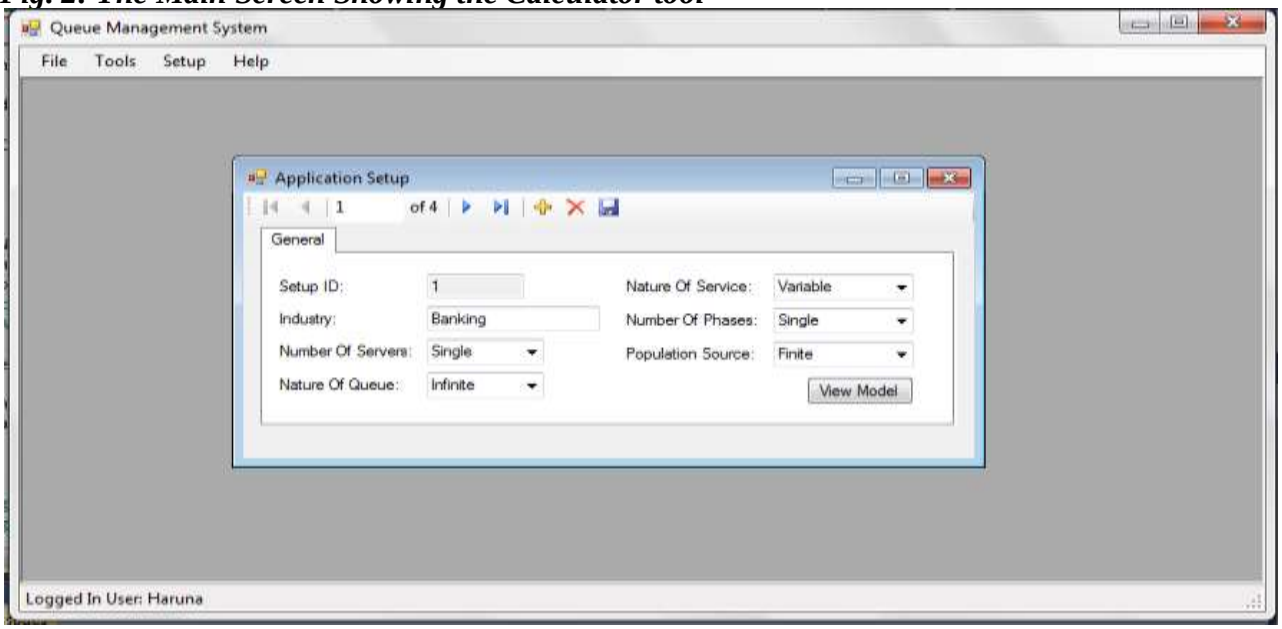


Fig.3: The Application Setup Module Interface

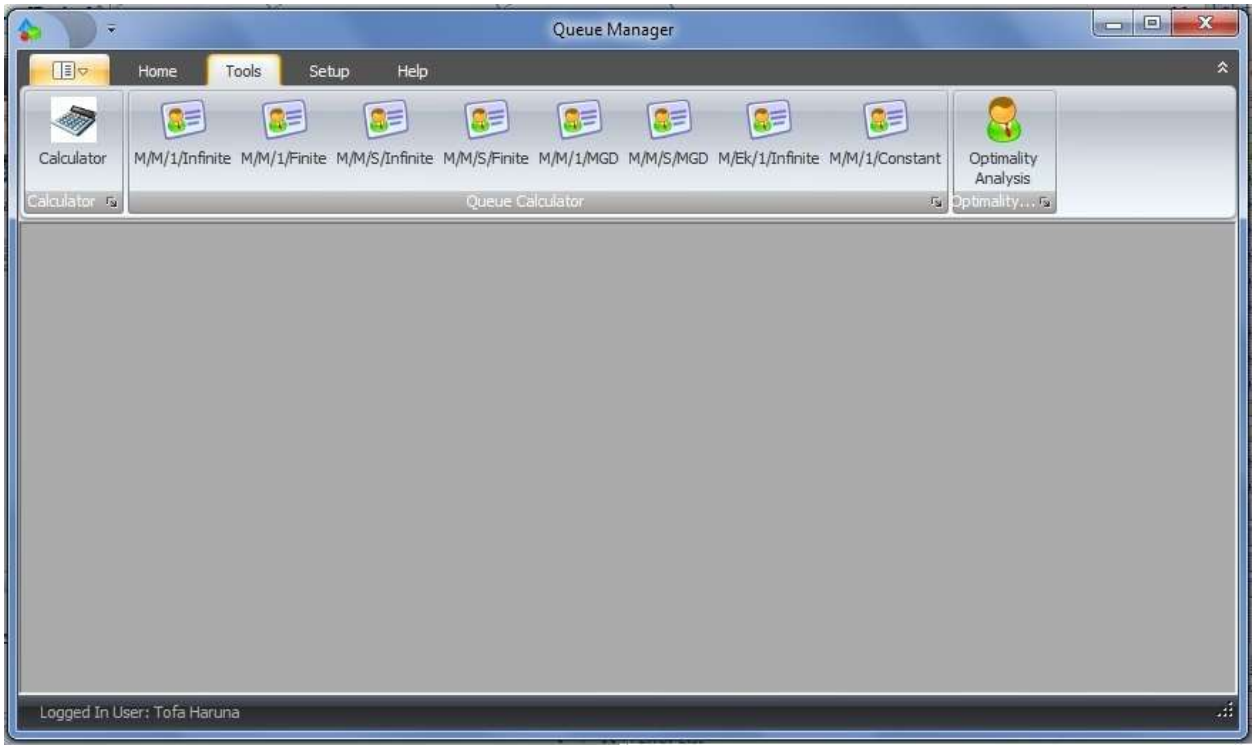


Fig. 4: The General Analysis Module Interface

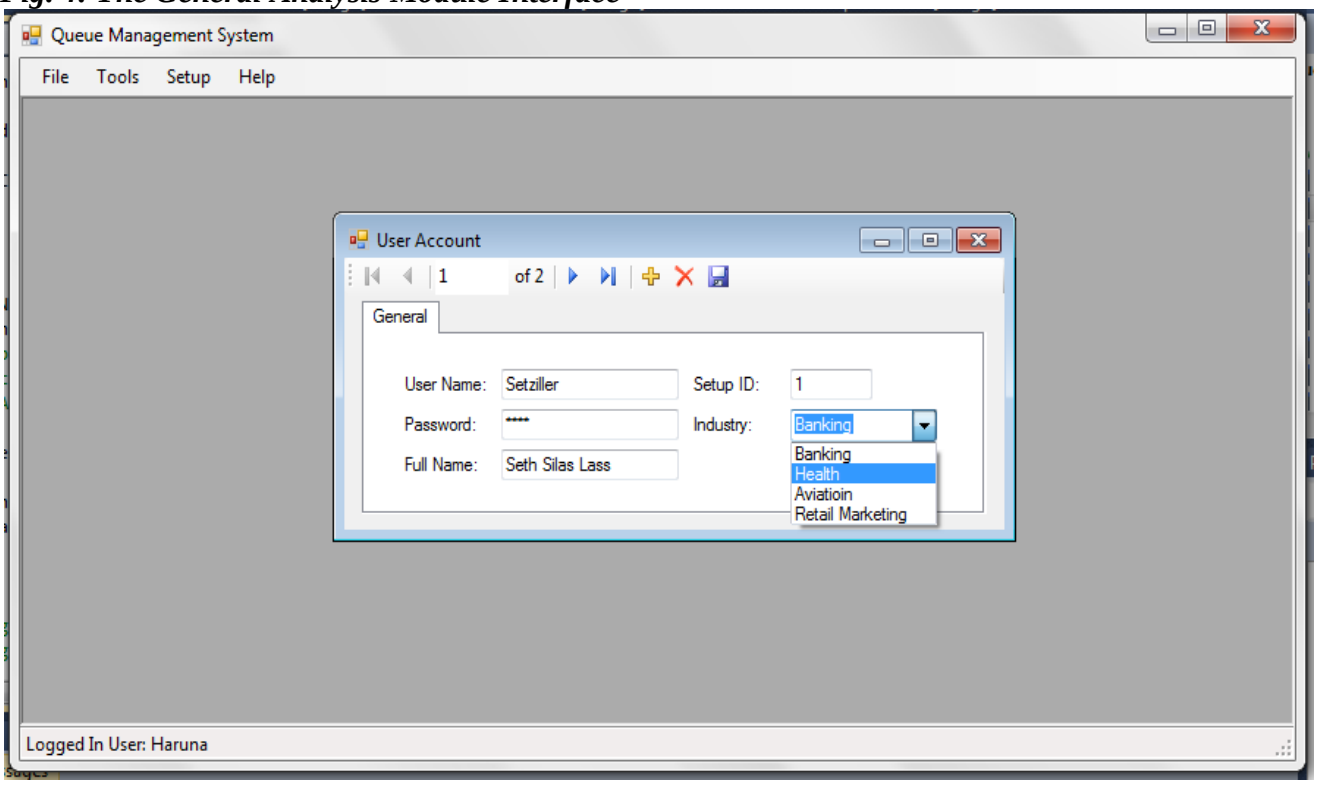


Fig.5: The User Account Setup Module Interface

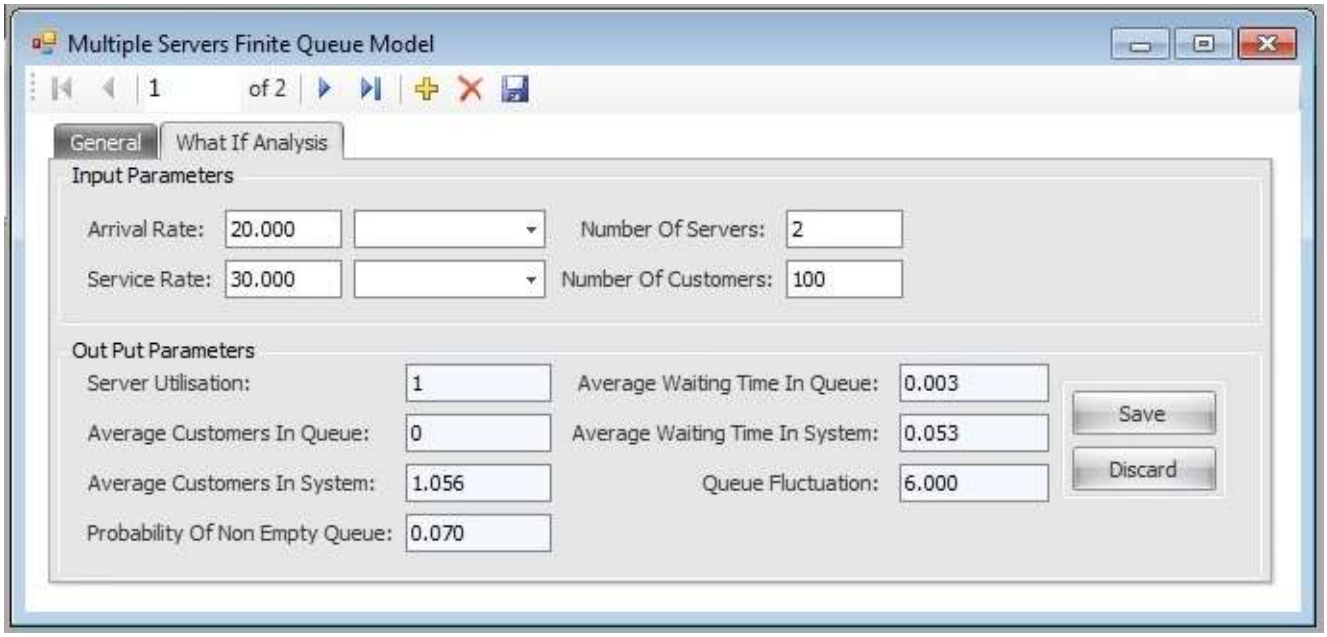


Fig.6: The What -If- Analysis Module Interface

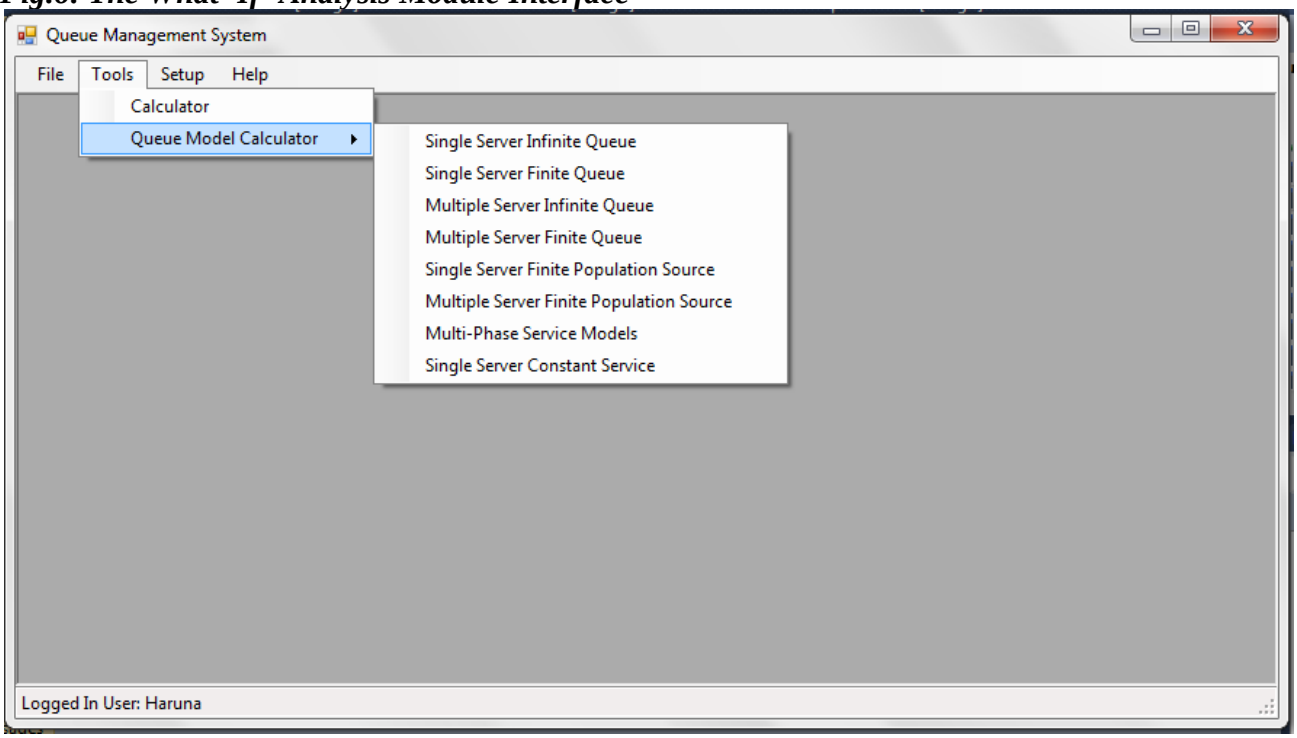


Fig.7: The Queue Model Calculator Module Interface

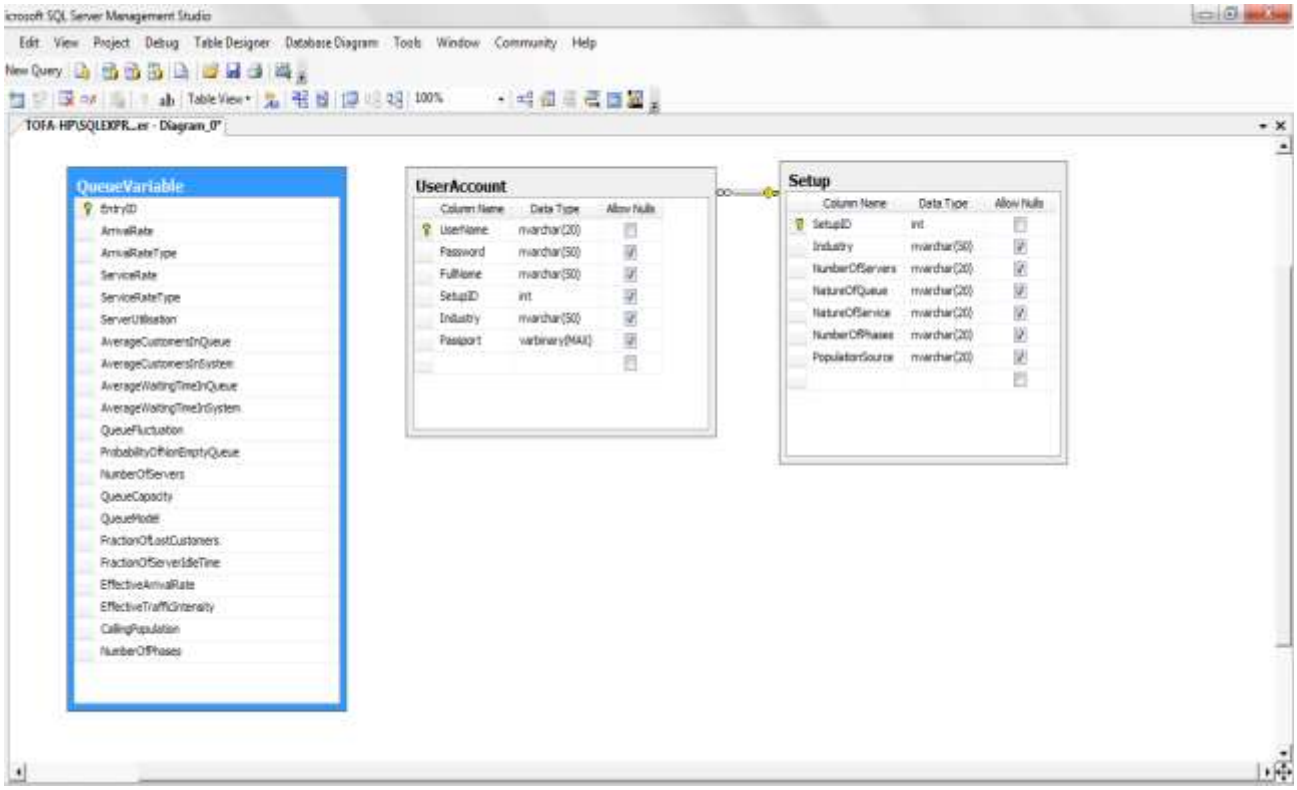


Fig.8: The System Database Diagram