**AN EVALUATION APPROACH FOR PARAMETER ESTIMATION OF SOFTWARE RELIABILITY GROWTH MODELS**

**A Major Project Report**

*Submitted in partial fulfilment of the*

*requirements for the award of the degree*

*of*

Bachelor of Technology

in

**INFORMATION TECHNOLOGY**

by

**1210711403 1210711410 1210711444 1210711454**



**DEPARTMENT OF INFORMATION TECHNOLOGY**

**GITAM INSTITUTE OF TECHNOLOGY**

**GITAM UNIVERSITY**

**VISAKHAPATNAM-530045, AP (INDIA)**

**April, 2015**

**CERTIFICATE**

I hereby certify that the work which is being presented in the B.Tech Major Project Report titled “**An Evaluation Approach For Parameter Estimation Of Software Reliability Growth Models**”, in partial fulfilment of the requirements for the award of the **Bachelor of Technology in INFORMATION TECHNOLOGY** and submitted to the Department of Information Technology of GITAM Institute of Technology, GITAM University, Visakhapatnam, A.P is an authentic record of my own work carried out during a period from September 2014 to March 2015 under the supervision **of Mr. B.Ravi Teja, Assistant Professor, IT Department**.

The matter submitted in this thesis has not been submitted by me for the award of any other degree elsewhere.

*Signature of Candidate*

**Anupama Das**

**Roll No. 1210711403**

This is to certify that the above statement made by the candidate is correct to the best of my knowledge.

*Signature of Supervisor*

Date**: Mr. B.Ravi Teja, Assistant Professor**

**Project Supervisor**

**DECLARATION**

I hereby declare that the project report titled “**AN EVALUATION APPROACH FOR PARAMETER ESTIMATION OF SOFTWARE RELIABILITY GROWTH MODELS**” is an original and authentic work done in the Department of Information Technology, GITAM Institute of Technology, GITAM University, Visakhapatnam, submitted in partial fulfilment of the requirements for the award of degree of Bachelor of Technology in Information Technology. I assure that this project is not submitted to any other university or educational institution.

Anupama Das

(1210711403)

**LIST OF FIGURES**

**DESCRIPTION PG NO.**

1. FIGURE 1- ABC FLOWCHART 15
2. FIGURE 2-GA FLOWCHART 18
3. FIGURE 3- LOGIN WINDOW 38
4. FIGURE 4- LOGIN WINDOW WITH DETAILS 39
5. FIGURE 5- NEW USER WINDOW 40
6. FIGURE 6- NEW USER WINDOW WITH DETAILS 41
7. FIGURE 7- SR GROWTH MODEL UI 42
8. FIGURE 8- SR GROWTH MODEL UI WITH DETAILS 43
9. FIGURE 9-FAILURE GRAPH 51
10. FIGURE 10-RELIABILITY GRAPH 52

**ACKNOWLEDGMENT**

The satisfaction that accompanies the successful completion of any task would be incomplete without the mention of people who made it possible and whose constant guidance and encouragement crown all the effort with success.

We express our deep sense of gratitude to **Mr.Ravi Teja Bhima , Assistant Professor**, Department of Information Technology, GIT, GITAM UNIVERSITY, Visakhapatnam who is our internal guide for continuously helping and giving us valuable suggestions to guide us in the successful completion of the project.

. It gives us boundless pleasure to avail ourselves this opportunity to express our deep sense of gratitude and wholehearted thanks to our professor **Dr.P.V.Lakshmi, Head of The Department**, Department of Information Technology, GIT, GITAM UNIVERSITY, Visakhapatnam who has continuously helped and gave valuable suggestions to guide us in the successful completion of the project. We are also grateful to **Dr.K.Lakshmi Prasad, PRINCIPAL**, GIT, GITAM University for his constant support in successful completion of our study. We are thankful to the project coordinator **Dr.G.Srinivasa Rao, Senior Assistant Professor**, Department of Information Technology, GIT, GITAM UNIVERSITY, Visakhapatnam for aiding us in the successful completion of the project.

We are thankful to the project coordinator **Dr.G.Srinivasa Rao, Senior Assistant Professor**, Department of Information Technology, GIT, GITAM UNIVERSITY, Visakhapatnam for aiding us in the successful completion of the project.We would like to thank **Mr.Mallikharjuna Rao, Assistant Professor** and other faculty members for their suggestions and support in the tenure of our project. We would like to thank our friends and family members for their valuable co-operation.

**ABSTRACT**

Software Reliability is the probability of failure free operation of software in a provided time period under specified conditions. Software testing is a process to detect faults in the totality and worth of developed computer software. A large effort is necessary in testing to increases the rate of detection and correction of error to increase the reliability of the software.

Software reliability engineering is an established area of software engineering research and practice that is concerned with the improvement and measurement of reliability. For the analysis typically stochastic software reliability models are used. They model the failure process of the software and use other software metrics or failure data as a basis for parameter estimation. The models are able to estimate the current reliability and to predict future failure behavior. Unreliability of any product comes due to the failures or presence of faults in the system. Measuring software reliability remains a difficult problem because we don't have a good understanding of the nature of software. We cannot find a suitable way to measure software reliability, and most of the aspects related to software reliability. In this work we have proposed an efficient software reliability growth model (SRGM). The model provides better failure rate recognition, better way to remove the faults and soon. In our work, we explore the use of Optimization algorithm to estimate SRGM parameters. The parameter estimation can help in better software reliability model. The optimization process employed in our proposed method is Artificial Bee Colony (ABC) and Genetic Algorithm (GA). The optimization of parameter improves the quality of parameter being selected for reliability growth model resulting in improved reliability growth. The implementation is done in the working platform of JAVA and eclipse using various software reliability parameters.

**TABLE OF CONTENTS**

Sr.No. Declaration PageNo

CERIFICATE II

DECLARATION III

LIST OF FIGURES IV

ACKNOWLEDGEMENT V

ABSTRACT VI

1. CHAPTER 1- INTRODUCTION 9
2. CHAPTER 2-LITERATURE SURVEY 14
3. CHAPTER 3-SYSTEM ANALYSIS AND DESIGN 19
   1. Software Requirements Specifications 20
      1. Functional Requirements 20
      2. Non functional Requirements 21
   2. System Requirements 22
      1. Hardware Requirements 23
      2. Software Requirements 23

1. CHAPTER 4- IMPLEMENTATION 24
   1. Pseudo Code 26

4.1.1 ABC Algorithm 33

4.1.2 GA Algorithm 34

* 1. Built-In Functions 35
  2. User Defined Function 36
  3. Implementation View 38

1. CHAPTER 5- TESTING 45
   1. Unit Testing 46
   2. Integration testing 47
   3. Test Cases 48

1. CHAPTER 6- RESULTS 50

7. CHAPTER 7-CONCLUSION 53

8. CHAPTER 8-REFERENCES 55

**CHAPTER 1**

**INTRODUCTION**

**INTRODUCTION**

Today, almost everyone in the world is directly or indirectly affected by computer systems: Computers are used in diverse areas for various applications including air traffic control, nuclear reactors, aircraft, real-time sensor networks, industrial process control, automotive mechanical and safety control, and hospital health care, affecting many millions of people. An application of computer systems to the hospital health care is the monitoring of heart patients. In hospitals so equipped, sensors that detect electrical signals associated with heart activity are attached to the patient’s heart area. The signals from these sensors are transmitted along wires to a computer programmed to analyze such data. If the incoming data indicate that the patient is doing well, the computer generated no output. If the data indicate the onset of serious conditions, the computer signals an alarm at the nursing station indicating which patient needs human care and the kind of help most apt to be useful. As the functionality of computer operations becomes more essential and yet more complicated and critical applications increase in size and complexity, there is a great need for looking at ways to quantify and predict the reliability of computer systems in various complex operating environments (Pham 2005c). Faults, especially with logic, in software design thus become more subtle. Usually logic errors in the software are not hard to fix but diagnosing logic bugs is the most challenging for many reasons. The fault again is usually subtle. Let us look at an example. A man wants to withdraw $50 at an automatic transfer machine (ATM) from a checking account held jointly with his wife. Almost simultaneously, at another machine, his wife also begins the deposit of $500. Both the husband’s and the wife’s ATM read the account balance of $100 from the memory at the bank’s central computer. While the first ATM (husband’s machine) subtracts the withdrawal, the second ATM adds the deposit. Because withdrawals often take slightly longer to process than deposits, the wife’s ATM records a new balance of $600 before her husband’s transaction is complete. His ATM, obviously not knowing that the old balance has been changed and in fact increased, records a wrong balance of $50, instead of the new balance which should be $550!

The terms such as “software error”, “fault” and “failure” (IEEE Std. 610.12,1990) is defined as a mental mistake made by the programmer or designer. A fault is the manifestation of that error in the code. A software failure is defined as the occurrence of an incorrect output as a result of an input value that is received with respect to the specification. Generally, software faults are more insidious and much more difficult to handle than physical defects. In theory, software can be error-free, and unlike hardware, does not degrade or wear out but it does deteriorate. The deterioration here, however, is not a function of time. Rather, it is a function of the results of changes made to the software during maintenance, through correcting latent defects, modifying the code to changing requirements and specifications, environments and applications, or improving software performance. All design faults are present from the time the software is installed in the computer. In principle, these faults could be removed completely, but in reality the goal of perfect software remains elusive (Friedman and Voas 1995). Computer programs, which vary for fairly critical applications between hundreds and millions of lines of code, can make the wrong decision because the particular inputs that triggered the problem were not tested and corrected during the testing phase. Such inputs may even have been misunderstood or unanticipated by the designer who either correctly programmed the wrong interpretation or failed to identify the problem. These situations and other such events have made it apparent that we must determine the reliability of the software systems before putting them into operation. Research on software reliability engineering has been conducted during the past three decades and numerous statistical models have been proposed for estimating software reliability (Pham 1999a, 2000a). Most existing models for predicting software reliability are based purely on the observation of software product failures where they require a considerable amount of failure data to obtain an accurate reliability prediction. Some other research efforts recently have developed reliability models addressing fault coverage, testing coverage, and imperfect debugging processes. In contrast, not many software practitioners, developers, or users utilize these models to evaluate software system reliability as they do not know how to select and apply them. A survey conducted in the late 1990s by the American Society for Quality reported that only 4% of the participants responded positively when asked if they could use a software reliability model. Many researchers are currently pursuing the development of statistical models, based on nonhomogeneous Poisson process, semi quasi renewal, time series, that can be used to evaluate the reliability of real-world software systems. To develop an application-practice software reliability model and be able to make sound judgments when using the model, one needs to understand how software is produced and tested, the types of errors, and how errors are introduced. Environmental factors can help us justify the usefulness of the model and its applicability in a user environment. In other words, these models would be valuable if practitioners, software developers and users could use the information about the software development process, incorporating the environmental factors, thus giving greater confidence in estimates based on small numbers of failure data.

Software testing is the process of executing a program to locate an error. A good test case is one that has a high probability of finding undiscovered error(s). It is impossible to continue testing the software until all faults are detected and removed as testing of all possible inputs would require millions of years! Therefore, failure probabilities must be inferred from testing a sample of all possible input states called the input space. In other words, input space is the set of all possible input states. Similarly, output space is the set of all possible output states for a given software and input space. It is generally very difficult to test exhaustively a large computer program because of problems with dimensionality. If the input space consists of a single unbounded variable, then an infinite number of input cases will be needed to provide an exhaustive test of the program. If the input space is bounded, but contains a large number of independent variables, then the number of input cases needed for an exhaustive test will tend to be impossibly large, even if one accepts the use of discretization for each input variable.

**CHAPTER 2**

**LITERATURE SURVEY**

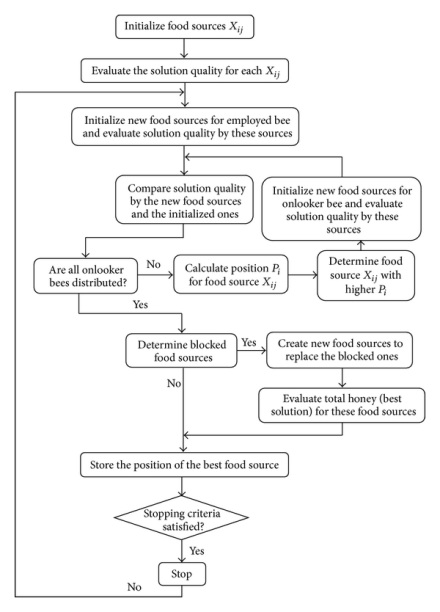
**LITERATURE SURVEY**

The project consists of three stages: running a java application and generating the test results. The failures are detected and then reduced on the application of population based optimizing algorithms, genetic algorithm and artificial bee colony algorithm. The number of failures is detected and the time interval is noted down during which the maximum number of failures have been reduced. The application is said to deliver optimal performance during that interval.

ARTIFICAL BEE COLONY ALGORITHM:

In the ABC model, the colony consists of three groups of bees: employed bees, onlookers and scouts. It is assumed that there is only one artificial employed bee for each food source. In other words, the number of employed bees in the colony is equal to the number of food sources around the hive. Employed bees go to their food source and come back to hive and dance on this area. The employed bee whose food source has been abandoned becomes a scout and starts to search for finding a new food source. Onlookers watch the dances of employed bees and choose food sources depending on dances.

In ABC, a population based algorithm, the position of a food source represents a possible solution to the optimization problem and the nectar amount of a food source corresponds to the quality (fitness) of the associated solution. The number of the employed bees is equal to the number of solutions in the population. At the first step, a randomly distributed initial population (food source positions) is generated. After initialization, the population is subjected to repeat the cycles of the search processes of the employed, onlooker, and scout bees, respectively. An employed bee produces a modification on the source position in her memory and discovers a new food source position. Provided that the nectar amount of the new one is higher than that of the previous source, the bee memorizes the new source position and forgets the old one. Otherwise she keeps the position of the one in her memory. After all employed bees complete the search process, they share the position information of the sources with the onlookers on the dance area. Each onlooker evaluates the nectar information taken from all employed bees and then chooses a food source depending on the nectar amounts of sources. As in the case of the employed bee, she produces a modification on the source position in her memory and checks its nectar amount. Providing that its nectar is higher than that of the previous one, the bee memorizes the new position and forgets the old one. The sources abandoned are determined and new sources are randomly produced to be replaced with the abandoned ones by artificial scouts.



*Fig1.ABC algorithm flowchart*

GENETIC ALGORITHM:

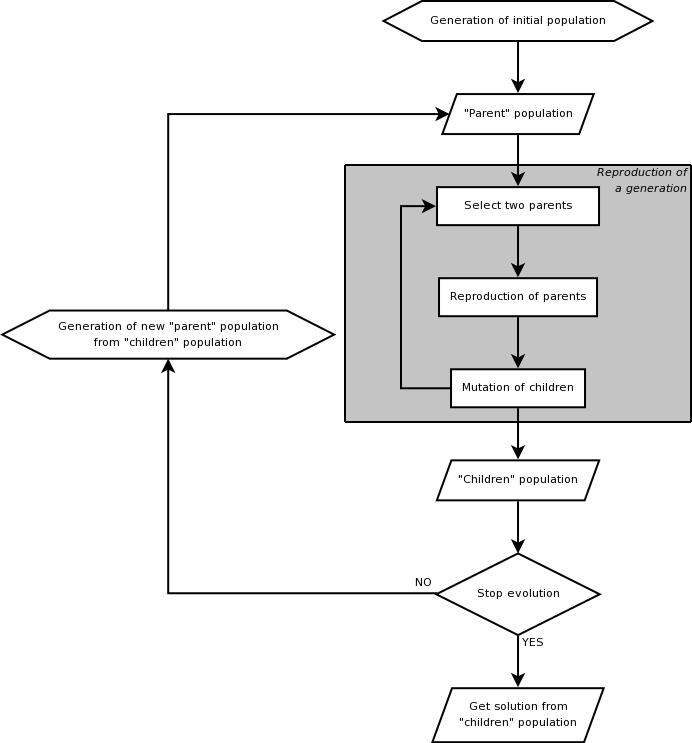
The GENETIC ALGORITHM is a model of machine learning which derives its behavior from metaphor of the processes of EVOLUTION in nature. This is done by the creation within a Machine of a POPULATION of INDIVIDUALs represented by CHROMOSOMEs, in essence a set of character strings that are analogous to the base-4 chromosomes that we see in our own DNA. The individuals in the population then go through a process of evolution. We should note that EVOLUTION (in nature or anywhere else) is not a purposive or directed process. That is, there is no evidence to support the assertion that the goal of evolution is to produce Mankind. Indeed, the processes of nature seem to boil down to different INDIVIDUALs competing for resources in the ENVIRONMENT. Some are better than others. Those that are better are more likely to survive and propagate their genetic material.

In nature, we see that the encoding for our genetic information (GENOME) is done in a way that admits asexual REPRODUCTION (such as by budding) typically results in OFFSPRING that are genetically identical to the PARENT. Sexual REPRODUCTION allows the creation of genetically radically different offspring that are still of the same general flavor (SPECIES). At the molecular level what occurs (wild over simplification alert!) is that a pair of CHROMOSOMEs bump into one another, exchange chunks of genetic information and drift apart. This is the RECOMBINATION peration, which GA/GPers generally refer to as CROSSOVER because of the way that genetic material crosses over from one chromosome to another. The CROSSOVER operation happens in an ENVIRONMENT where the SELECTION of who gets to mate is a function of the FITNESS of the INDIVIDUAL, i.e. how good the individual is at competing in its environment. Some GENETIC ALGORITHMs use a simple function of the fitness measure to select individuals (probabilistically) to undergo genetic operations such as crossover or asexual REPRODUCTION (the propagation of genetic material unaltered). This is fitness-proportionate selection. Other implementations use a model in which certain randomly selected individuals in a subgroup compete and the fittest is selected. This is called tournament selection and is the form of selection we see in nature when stags rut to vie for the privilege of mating with a herd of hinds. The two processes that most contribute to EVOLUTION are crossover and fitness based selection/reproduction.

As it turns out, there are mathematical proofs that indicate that the process of FITNESS proportionate REPRODUCTION is, in fact, near optimal in some senses. MUTATION also plays a role in this process, although how important its role is continues to ba a matter of debate (some refer to it as a backgroud operator, while others view it as playing the dominant role in the evolutionary process). It cannot be stressed too strongly that the GENETIC ALGORITHM (as a SIMULATION of a genetic process) is not a random search for a solution to a problem (highly fit INDIVIDUAL). The genetic algorithm uses stochastic processes, but the result is distinctly non-random (better than random). GENETIC ALGORITHMs are used for a number of different application areas. An example of this would be multidimensional OPTIMIZATION problems in which the character string of the CHROMOSOME can be used to encode the values for the different parameters being optimized. In practice, therefore, we can implement this genetic model of computation by having arrays of bits or characters to represent the CHROMOSOMEs. Simple bit manipulation operations allow the implementation of CROSSOVER, MUTATION and other operations. Although a substantial amount of research has been performed on variable- length strings and other structures, the majority of work with

GENETIC ALGORITHMs is focussed on fixed-length character strings. We should focus on both this aspect of fixed-lengthness and the need to encode the representation of the solution being sought as a character string, since these are crucial aspects that distinguish GENETIC PROGRAMMING, which does not have a fixed length representation and there is typically no encoding of the problem. When the GENETIC ALGORITHM is implemented it is usually done in a manner that involves the following cycle: Evaluate the FITNESS of all of the INDIVIDUALs in the POPULATION. Create a new population by performing operations such as CROSSOVER, fitness-proportionate REPRODUCTION and MUTATION on the individuals whose fitness has just been measured. Discard the old population and iterate using the new population.

One iteration of this loop is referred to as a GENERATION. There is no theoretical reason for this as an implementation model. Indeed, we do not see this punctuated behavior in POPULATIONs in nature as a whole, but it is a convenient implementation model. The first GENERATION (generation 0) of this process operates on a POPULATION of randomly generated INDIVIDUALs. From there on, the genetic operations, in concert with the FITNESS measure, operate to improve the population.



*Fig 2. GA algorithm flowchart*

**CHAPTER 3**

**SYSTEM ANALYSIS AND DESIGN**

**SYSTEM ANALYSIS AND DESIGN**

**3.1 SOFTWARE REQUIREMENTS SPECIFICATIONS**

A software requirement specification (SRS) is a description of a software system to be developed, laying out functional and non-functional requirements, and may include a set of use cases that describe interactions the users will have with the software.

Software requirements specification establishes the basis for an agreement between customers and contractors or suppliers (in market-driven projects, these roles may be played by the marketing and development divisions) on what the software product is to do as well as what it is not expected to do. Software requirements specification permits a rigorous assessment of requirements before design can begin and reduces later redesign. It should also provide a realistic basis for estimating product costs, risks, and schedules.

The software requirements specification document enlists enough and necessary requirements that are required for the project development. To derive the requirements we need to have clear and thorough understanding of the products to be developed or being developed. This is achieved and refined with detailed and continuous communications with the project team and customer till the completion of the software.

**3.1.1 FUNCTIONAL REQUIREMENTS**

In systems engineering a functional specification is a document that clearly and accurately describes the essential technical requirements for items, materials, or services including the procedures by which it can be determined that the requirements have been met. Specifications help avoid duplication and inconsistencies, allow for accurate estimates of necessary work and resources, act as a negotiation and reference document for engineering changes, provide documentation of configuration, and allow for consistent communication among those responsible for the eight primary functions of Systems Engineering. They provide a precise idea of the problem to be solved so that they can efficiently design the system and estimate the cost of design alternatives. They provide guidance to testers for verification (qualification) of each technical requirement.

A functional specification does not define the inner workings of the proposed system; it does not include the specification of how the system function will be implemented. Instead, it focuses on what various outside agents (people using the program, computer peripherals, or other computers, for example) might "observe" when interacting with the system. A typical functional specification might state as follows:

When the user clicks the OK button, the dialog is closed and the focus is returned to the main window in the state it was in before this dialog was displayed.

Such a requirement describes an interaction between an external agent (the user) and the software system. When the user provides input to the system by clicking the OK button, the program responds (or should respond) by closing the dialog window containing the OK button.

It can be informal, in which case it can be considered as a blueprint or user manual from a developer point of view, or formal, in which case it has a definite meaning defined in mathematical or programmatic terms. In practice, most successful specifications are written to understand and fine-tune applications that were already well-developed, although safety-critical software systems are often carefully specified prior to application development. Specifications are most important for external interfaces that must remain stable.

Functional requirements for the project:

• Objective: to quantify and predict the reliability of the bank application before and after the optimization.

• Input: The generated test results obtained after running the bank application.

• Output: The reliability is indicated through graphs and the efficiency of the optimizing algorithm is known.

**3.1.2 NON FUNCTIONAL REQUIREMENTS**

In systems engineering and requirements engineering, a non-functional requirement is a requirement that specifies criteria that can be used to judge the operation of a system, rather than specific behaviours. This should be contrasted with functional requirements that define specific behaviour or functions. The plan for implementing functional requirements is detailed in the system design. The plan for implementing non-functional requirements is detailed in the system architecture.

Broadly, functional requirements define what a system is supposed to do and non-functional requirements define how a system is supposed to be. Functional requirements are usually in the form of "system shall do <requirement>", an individual action of part of the system, perhaps explicitly in the sense of a mathematical function, a black box description input, output, process and control functional model or IPO Model. In contrast, non-functional requirements are in the form of "system shall be <requirement>", an overall property of the system as a whole or of a particular aspect and not a specific function. The systems' overall properties commonly mark the difference between whether the development project has succeeded or failed.

Non-functional requirements are often called qualities of a system. Other terms for non-functional requirements are "constraints", "quality attributes", "quality goals", "quality of service requirements" and "non-behavioural requirements". Informally these are sometimes called the "utilities", from attributes like stability and portability. Qualities, that are non-functional requirements, can be divided into two main categories:

• Execution qualities, such as security and usability, which are observable at run time.

• Evolution qualities, such as testability, maintainability, extensibility and scalability, which are embodied in the static structure of the software system.

**3.2 SYSTEM REQUIREMENTS**

To be used efficiently, all computer software needs certain hardware components or other software resources to be present on a computer.[1] These prerequisites are known as (computer) system requirements and are often used as a guideline as opposed to an absolute rule. Most software defines two sets of system requirements: minimum and recommended. With increasing demand for higher processing power and resources in newer versions of software, system requirements tend to increase over time. Industry analysts suggest that this trend plays a bigger part in driving upgrades to existing computer systems than technological advancements. A second meaning of the term of System requirements, is a generalisation of this first definition, giving the requirements to be met in the design of a system or sub-system. Typically an organisation starts with a set of Business requirements and then derives the System requirements from there.

Often manufacturers of games will provide the consumer with a set of requirements that are different from those that are needed to run a software. These requirements are usually called the Recommended Requirements. These requirements are almost always of a significantly higher level than the minimum requirements, and represent the ideal situation in which to run the software. Generally speaking this is a better guideline than minimum system requirements in order to have a fully usable and enjoyable experience with a software**.**

**3.2.1 HARDWARE REQUIREMENTS**

To be used efficiently, all computer software needs certain hardware components or other software resources to be present on a computer. These prerequisites are known as (computer) system requirements and are often used as a guideline as opposed to an absolute rule. Most software defines two sets of system requirements: minimum and recommended. With increasing demand for higher processing power and resources in newer versions of software, system requirements tend to increase over time. Industry analysts suggest that this trend plays a bigger part in driving upgrades to existing computer systems than technological advancements. A second meaning of the term of System requirements, is a generalisation of this first definition, giving the requirements to be met in the design of a system or sub-system. Typically an organisation starts with a set of Business requirements and then derives the System requirements from there.

Hardware requirement specifications for the project are:

• VDU: Monitor

• Input Devices: Keyboard and mouse

• RAM: 512 MB

• Processor: P4 or above

• Storage: less than 100 MB of HDD space

**3.2.2 SOFTWARE REQUIREMENTS**

Software requirements deal with defining software resource requirements and prerequisites that need to be installed on a computer to provide optimal functioning of an application. These requirements or prerequisites are generally not included in the software installation package and need to be installed separately before the software is installed.

Software requirements for the project are:

• Operating system: Any operating system

• Run Time: OS compatible JVM

• Other softwares: MS Access , Eclipse

**CHAPTER 4**

**IMPLEMENTATION**

**4.1 PSEUDO CODE**

import java.awt.\*;

import java.awt.event.\*;

import javax.swing.\*;

class GuiAccTest extends Frame implements ActionListener

{

Label lab=new Label(" ");

Label lab1=new Label(" ");

TextField t[]=new TextField [4];

Label l[]=new Label [4];

Button but=new Button("Create Account");

Button but1=new Button("Test Account");

BankAccount b;

GuiAccTest()

{

addWindowListener(new NewWindowAdapter());

setLayout(new GridLayout(2,0));

Panel p=new Panel();

Panel p1=new Panel();

but.addActionListener(this);

but1.addActionListener(this);

p.setLayout(new GridLayout(5,2));

p1.add(lab1);

p1.add(lab);

l[0]=new Label("Account Number");

l[1]=new Label("Initial Balance");

l[2]=new Label("Deposit Amount");

l[3]=new Label("Withdraw Amount");

for(int i=0;i<4;i++)

{

t[i]=new TextField(10);

p.add(l[i]);

p.add(t[i]);

}

p.add(but);

p.add(but1);

but1.setVisible(false);

l[2].setVisible(false);

l[3].setVisible(false);

t[2].setVisible(false);

t[3].setVisible(false);

add(p);

add(p1);

}

String testAccount(int d\_amt,int w\_amt)

{

String msg;

b.deposit(d\_amt);

msg="Transaction Succesful";

try

{

b.withdraw(w\_amt);

}catch(FundsInsufficientException fe)

{

fe=new FundsInsufficientException(b.amount,w\_amt);

msg=String.valueOf(fe);

}

return msg;

}

public void actionPerformed(ActionEvent ae)

{

String str=ae.getActionCommand();

if(str.equals("Create Account"))

{

b=new BankAccount(Integer.parseInt(t[0].getText()),Integer.parseInt(t[1].getText()));

but1.setVisible(true);

l[2].setVisible(true);

l[3].setVisible(true);

t[2].setVisible(true);

t[3].setVisible(true);

but.setVisible(false);

l[0].setVisible(false);

l[1].setVisible(false);

t[0].setVisible(false);

t[1].setVisible(false);

lab1.setText("Account : "+b.accnum+", Current Balance : "+b.amount);

return;

}

else

{

lab.setText(testAccount(Integer.parseInt(t[2].getText()),Integer.parseInt(t[3].getText())));

lab1.setText("Account : "+b.accnum+", Current Balance : "+b.amount);

}

}

public static void main(String arg[])

{

GuiAccTest at=new GuiAccTest();

at.setTitle("Bank Account Tester");

at.setSize(600,200);

at.setVisible(true);

}

}

class NewWindowAdapter extends WindowAdapter

{

public void windowClosing(WindowEvent we)

{

System.exit(0);

}

}

class BankAccount

{

int accnum;

int amount;

BankAccount(int num,int amt)

{

accnum=num;

amount=amt;

}

public void deposit(int amt)

{

amount=amount+amt;

}

public void withdraw(int amt) throws FundsInsufficientException

{

if(amt>amount)

throw new FundsInsufficientException(amount,amt);

else

amount=amount-amt;

}

}

class FundsInsufficientException extends Exception

{

int balance;

int withdraw\_amount;

FundsInsufficientException(int bal,int w\_amt)

{

balance=bal;

withdraw\_amount=w\_amt;

}

public String toString()

{

return "Your withdraw amount ("+withdraw\_amount+") is less than the balance ("+balance+"). No withdrawal was recorded.";

}

}

**4.1.1 ARTIFICIAL BEE COLONY ALGORITHM**

The main steps of the algorithm are given below:

• Initial food sources are produced for all employed bees

REPEAT

• Each employed bee goes to a food source in her memory and determines a neighbour source, then evaluates its nectar amount and dances in the hive

• Each onlooker watches the dance of employed bees and chooses one of their sources depending on the dances, and then goes to that source. After choosing a neighbour around that, she evaluates its nectar amount.

• Abandoned food sources are determined and are replaced with the new food sources discovered by scouts.

• The best food source found so far is registered.

UNTIL (requirements are met)

**4.1.2 GENETIC ALGORITHM**

The main steps of algorithm are

// start with an initial time

t := 0;

// initialize a usually random population of individuals

initpopulation P (t);

// evaluate fitness of all initial individuals of population

evaluate P (t);

// test for termination criterion (time, fitness, etc.)

while not done do

// increase the time counter

t := t + 1;

// select a sub-population for offspring production

P' := selectparents P (t);

// recombine the "genes" of selected parents

recombine P' (t);

// perturb the mated population stochastically

mutate P' (t);

// evaluate it's new fitness

evaluate P' (t);

// select the survivors from actual fitness

P := survive P,P' (t);

Od

end GA.

**4.2 BUILT IN FUNCTIONS**

The built in functions used in the making of the project are listed as follows:

• Set forceground()

• Set background()

• Set bounce()

• Add()

• Set visible()

• Set size()

• Set editable()

• Set font()

• Try()

• Catch()

• Throw()

• Dispose()

• newjlabel()

• newjtextfield()

• keypressed()

• keyreleased()

• keytyped()

**4.3 USER DEFINED FUNCTIONS**

• ArtificialBeecolony()

Syntax: public ArtificialBeeColony (int n)

• SendEmplyedBees()

Syntax: public void sendEmployedBees()

• SendOnlookerBee()

Syntax: public void sendOnlookerBees()

• SendToWork()

Syntax: public void sendToWork(Honey currentBee, Honey neighborBee)

• SendScoutbees()

Syntax: public void sendScoutBees()

• GetFitness()

Syntax: public void getFitness()

• CalculateProbabilities()

Syntax: public void calculateProbabilities()

• Get Random Numbers()

Syntax: public int getRandomNumber(int low, int high)

• Randomly Arrange()

Syntax: public void randomlyArrange(int index)

• MemorizeBestFoodsources()

Syntax: public void memorizeBestFoodSource()

• TesterABC()

Syntax: public TesterABC()

• Test()

Syntax: public void test(int maxLength, int trialLimit, int maxEpoch)

• Log Parameters()

Syntax: public void logParameters()

• Print Runtimes()

Syntax: public void printRuntimes()

• Writer()

Syntax: publicWriter()

• Add()

Syntax: public void add(String line)

• Clear Board()

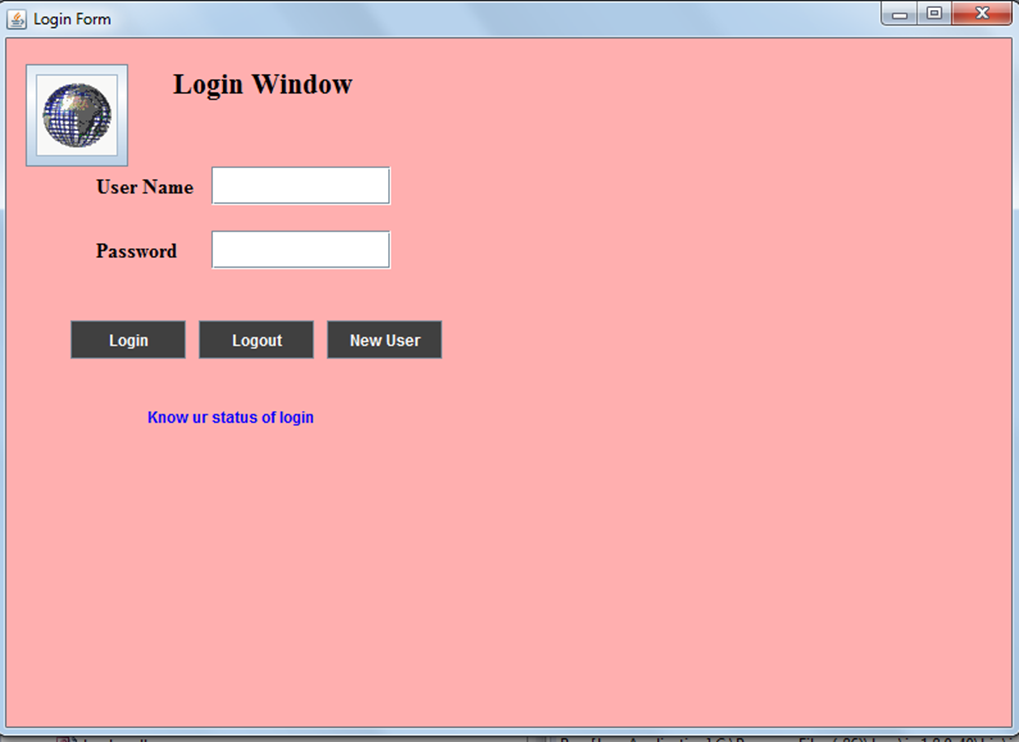
Syntax: public void clearBoard(String[][] board, int n)

• Print Board()

Syntax: public void printBoard(String[][] board, int n)

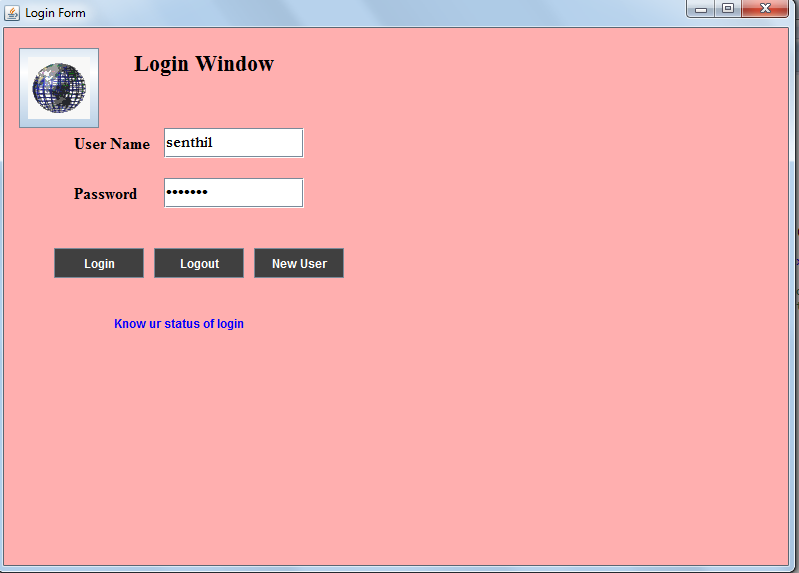
**4.4 IMPLEMENTATION VIEW**

**LOGIN WINDOW:**

****

*Fig3 Login window*

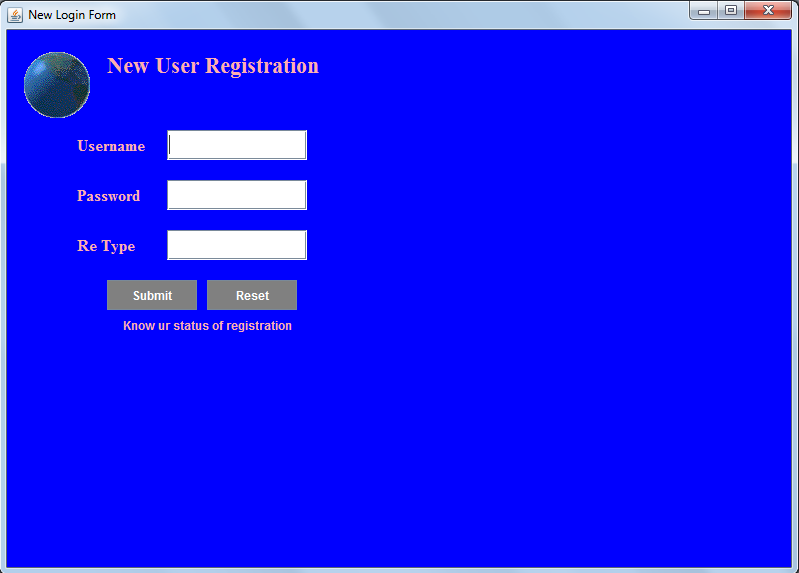
This is the login window which pops up on running the bank application. It acts as an interface between the database stored in the bank and the users. It asks for the username and password to check the authenticity of the login user.



*Fig 4 Login window with details*

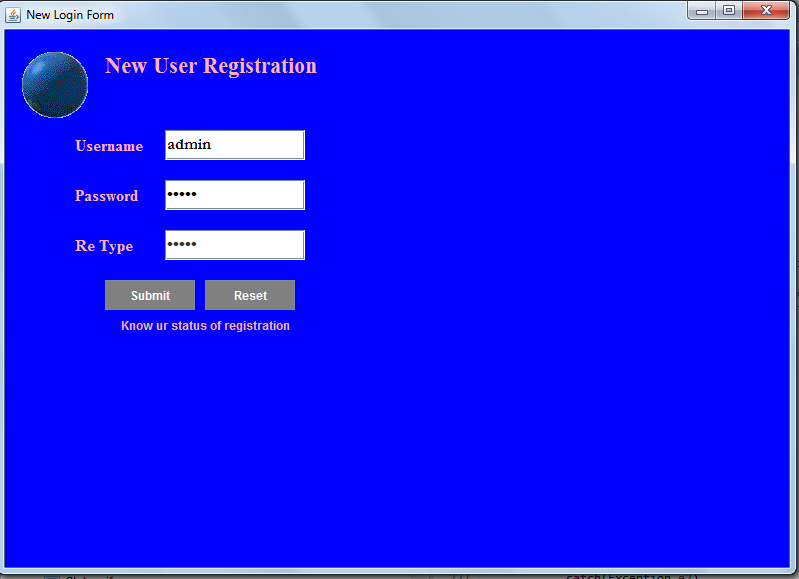
If the details entered are authentic , user gets logged in and accesses the bank application.

**NEW USER:**



*Fig5 New User*

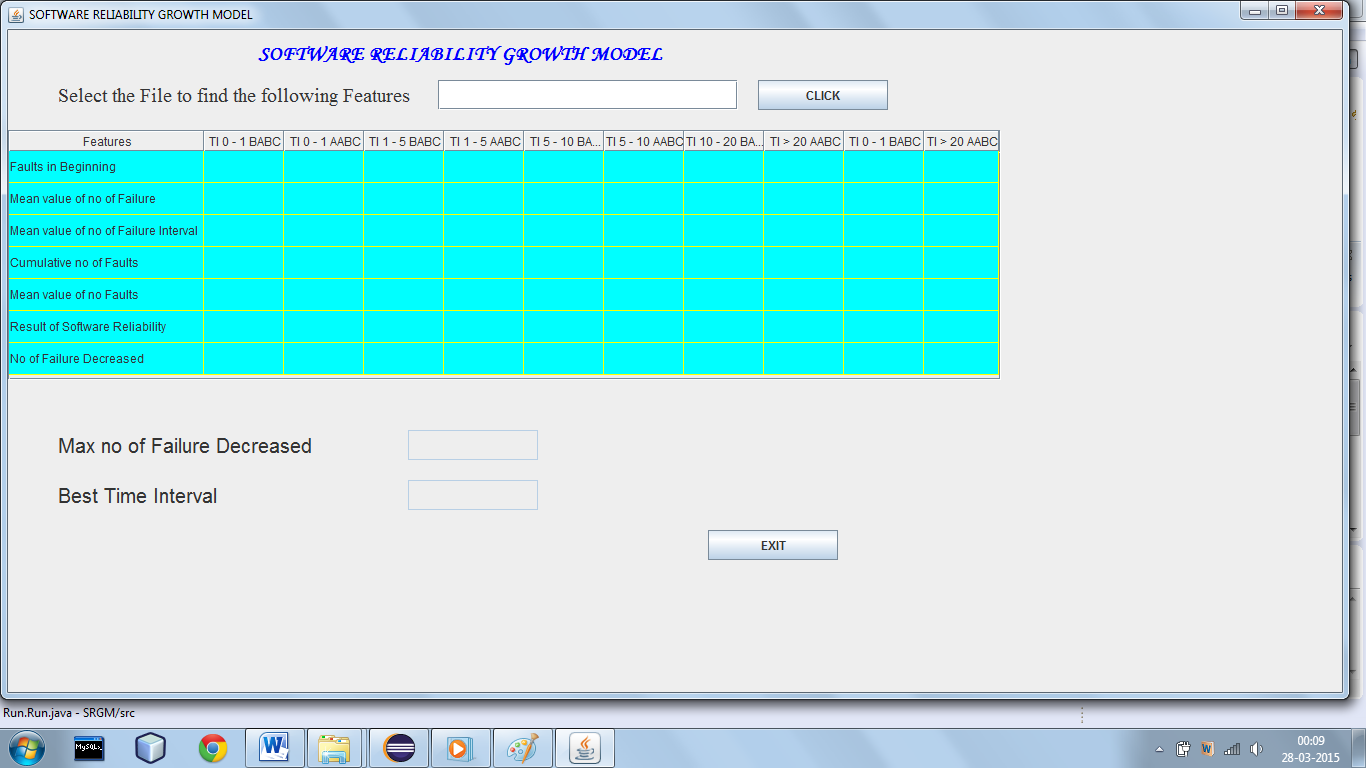
This window indicates the result of the third button “new user” in the previous window. The username , password are entered by the user in the textboxes provided . To avoid the mistakes of typing, the retype textfield is given to re-enter the password.



*Fig 6 New User with details*

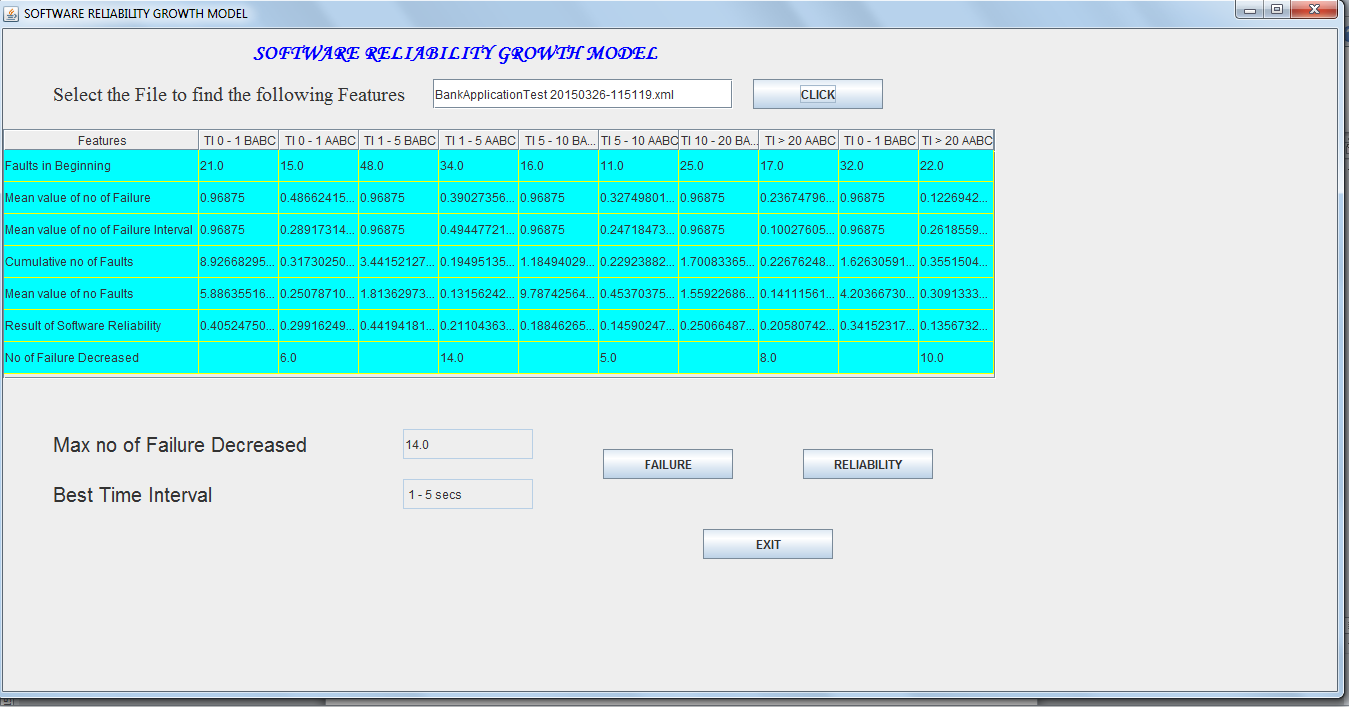
To register, the submit button should be pressed.The reset button is given for entry of the details of the new user.

**SR GROWTH MODELS:**



*Fig 7 SR Growth model UI*

The above window describes the software reliability growth models. These models measures and predicts the improvement of reliability programs through the testing process. The growth model represents the reliability or failure rate of a system as a function of time or the number of test cases.



*Fig 8 SR Growth Model UI with details*

The above picture gives the details on the number of failure and their occurrence in specific time intervals. From the results it is evident that the failures are decreased after the application of optimizing algorithm and the best time interval is indicate for which the maximum number of failures were reduced.

**CHAPTER 5**

**TESTING**

**TESTING**

Software testing is an investigation conducted to provide stakeholders the information about the quality of the product or service under test. Software testing can also provide an objective, independent view of the software to allow the business to appreciate and understand the risks of software implementation. Test techniques include, but are not limited to, the process of executing a program or application with the intent of finding software bugs (errors or other defects). It involves the execution of a software component or system component to valuate one or more properties of interest. In general, these properties indicate the extent to which the component or system under test: meets the requirements that guided its design and development, responds correctly to all kinds of inputs, performs its functions within an acceptable time, is sufficiently usable, can be installed and run in its intended environments, and achieves the general result its stakeholders desire. As the number of possible tests for even simple software components is practically infinite, all software testing uses some strategy to select tests that are feasible for the available time and resources. As a result, software testing typically (but not exclusively) attempts to execute a program or application with the intent of finding software bugs (errors or other defects). Software testing can provide objective, independent information about the quality of software and risk of its failure to users and/or sponsors.

Software testing can be conducted as soon as executable software (even if partially complete) exists. The overall approach to software development often determines when and how testing is conducted. For example, in a phased process, most testing occurs after system requirements have been defined and then implemented in testable programs. In contrast, under an Agile approach, requirements, programming, and testing are often done concurrently.

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub-assemblies, assemblies and/or a finished product. It is the process of exercising software with the intent of ensuring that the software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of test. Each test type addresses a specific testing requirement.

Test objectives

• All field entries must work properly.

• Pages must be activated from the identified link.

The entry screen, messages and responses must not be delayed

**5.1 Testing levels**

There are generally four recognized levels of tests: unit testing, integration testing, system testing, and acceptance testing. Tests are frequently grouped by where they are added in the software development process, or by the level of specificity of the test. The main levels during the development process as defined by the SWEBOK guide are unit, integration, and system testing that are distinguished by the test target without implying a specific process model. Other test levels are classified by the testing objective.

**5.1.1 Unit testing**

Unit testing, also known as component testing, refers to tests that verify the functionality of a specific section of code, usually at the function level. In an object oriented environment, this is usually at the class level, and the minimal unit tests include the constructors and destructors. These types of tests are usually written by developers as they work on code (white box style), to ensure that the specific function is working as expected. One function might have multiple tests, to catch corner cases or other branches in the code. Unit testing alone cannot verify the functionality of a piece software, but rather is used to ensure that the building blocks of the software work independently from each other.

Unit testing is a software development process that involves synchronized application of a broad spectrum of defect prevention and detection strategies in order to reduce software development risks, time, and costs. It is performed by the software developer or engineer during the construction phase of the software development lifecycle. Rather than replace traditional QA focuses, it augments it. Unit testing aims to eliminate construction errors before code is promoted to QA; this strategy is intended to increase the quality of the resulting software as well as the efficiency of the overall development and QA process.

Depending on the organization's expectations for software development, unit testing might include static code analysis, data flow analysis, metrics analysis, peer code reviews, code coverage analysis and other software verification practices.

**5.1.2 Integration testing**

Integration testing is any type of software testing that seeks to verify the interfaces between components against a software design. Software components may be integrated in an iterative way or all together ("big bang"). Normally the former is considered a better practice since it allows interface issues to be located more quickly and fixed.

Integration testing works to expose defects in the interfaces and interaction between integrated components (modules). Progressively larger groups of tested software components corresponding to elements of the architectural design are integrated and tested until the software works as a system.

**5.1.3 System testing**

System testing, or end to end testing, tests a completely integrated system to verify that it meets its requirements. For example, a system test might involve testing a logon interface, then creating and editing an entry, plus sending or printing results, followed by summary processing or deletion (or archiving) of entries, then logoff. In addition, the software testing should ensure that the program, as well as working as expected, does not also destroy or partially corrupt its operating environment or cause other processes within that environment to become inoperative (this includes not corrupting shared memory, not consuming or locking up excessive resources and leaving any parallel processes unharmed by its presence).

**5.2 TEST CASES**

A test case, in software engineering is a set of conditions under which a tester will determine whether an application or software system or one of its features is working as it was originally established for.

The design of the test cases used in the project is

|  |  |
| --- | --- |
| Test Case Id | 1 |
| Objective :Login Window | |
| Description:  Step 1: Enter the user name.  Step 2: Enter the password.  Step 3: Click on login.  Step 4: Click on log out.  Step 5: Click on new user. | |
| Actions: No clashes occurred. | |
| Expected results : A message "the solution is accepted" is displayed | |
| Pass : Yes Fail : NIL | |

|  |  |
| --- | --- |
| Test Case Id | 2 |
| Objective : New User Registration. | |
| Description:  Step 1: Enter the username.  Step 2: Enter the password.  Step 3: Retype the password.  Step 4: Click on the submit.  Step 5: Click on the Reset. | |
| Actions: No clashes occurred. | |
| Expected results : A message "the solution is accepted" is displayed | |
| Pass : Yes Fail : NIL | |

|  |  |
| --- | --- |
| Test Case Id | RUN: 3 |
| Objective : Deposit form | |
| Description:  Step 1: Enter the account number  Step 2: Click on Search.  Step 3: Enter the amount to deposit.  Step 4: Click on the deposit.  Step 5: Clear data.  Step 6: Click on clear.  Step 7: Click on close. | |
| Actions: No clashes occurred.0 | |
| Expected results : A message "the solution is accepted" is displayed | |
| Pass : Yes Fail : NIL | |

|  |  |
| --- | --- |
| Test Case Id | 4 |
| Objective: Software reliability growth models. | |
| Description:  Step 1: Select the file to find the following features.  Step 2: Click on click.  Step 3: Enter the maximum no of failure decreased.  Step 4: Enter the best time interval.  Step 5: Click on the failure. | |
| Actions: No clashes occurred. | |
| Expected results : A message "the solution is accepted" is displayed | |
| Pass : Yes Fail : NIL | |

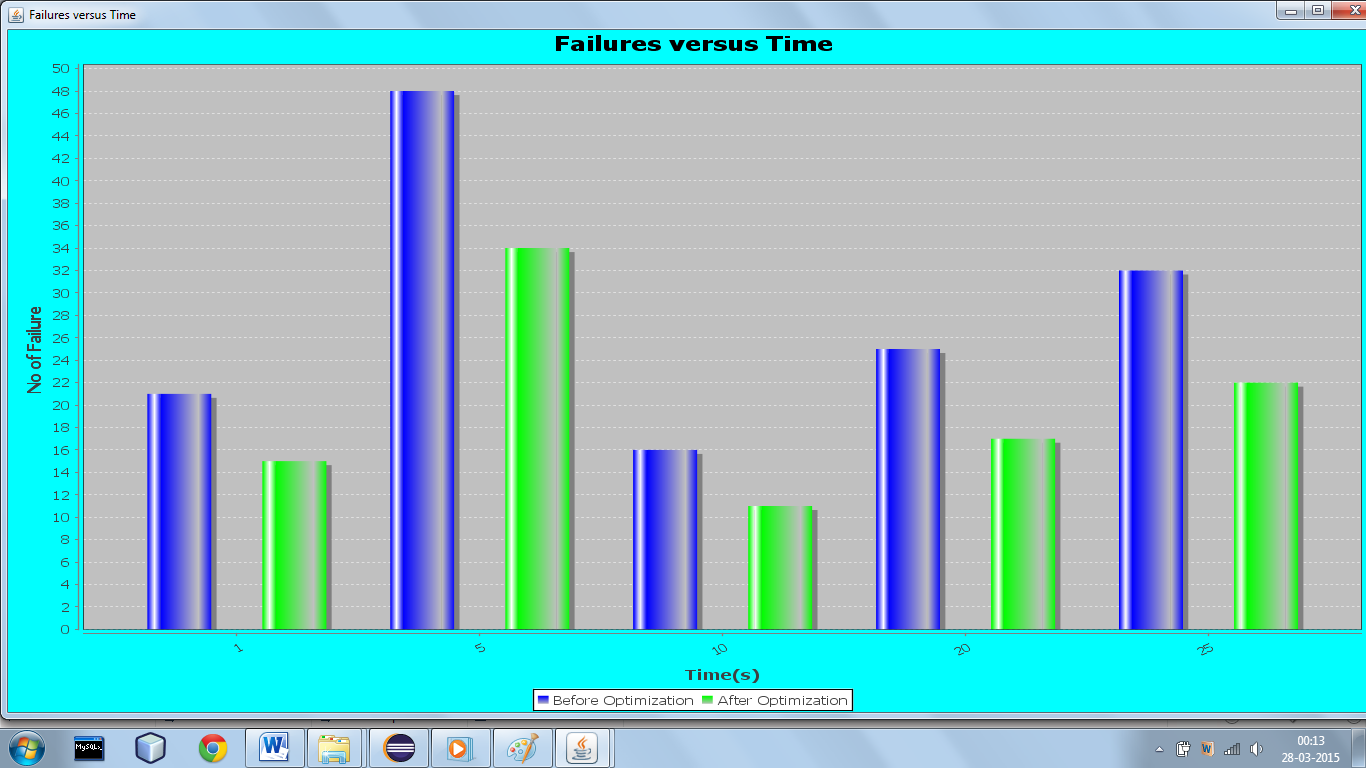
**CHAPTER 6**

**RESULT**

**RESULT**

Traditionally there are two common types of failure data: time domain data and interval domain data. These data are usually used by practitioners when analysing and predicting reliability applications. Some software reliability models can handle both types of data. The time domain approach involves recording the individual times at which failure occurred. The interval domain approach is characterized by counting the number of failures occurring during a fixed period. Using this method, the collected data are a count of the number of failures in the interval as illustrated in the fig.

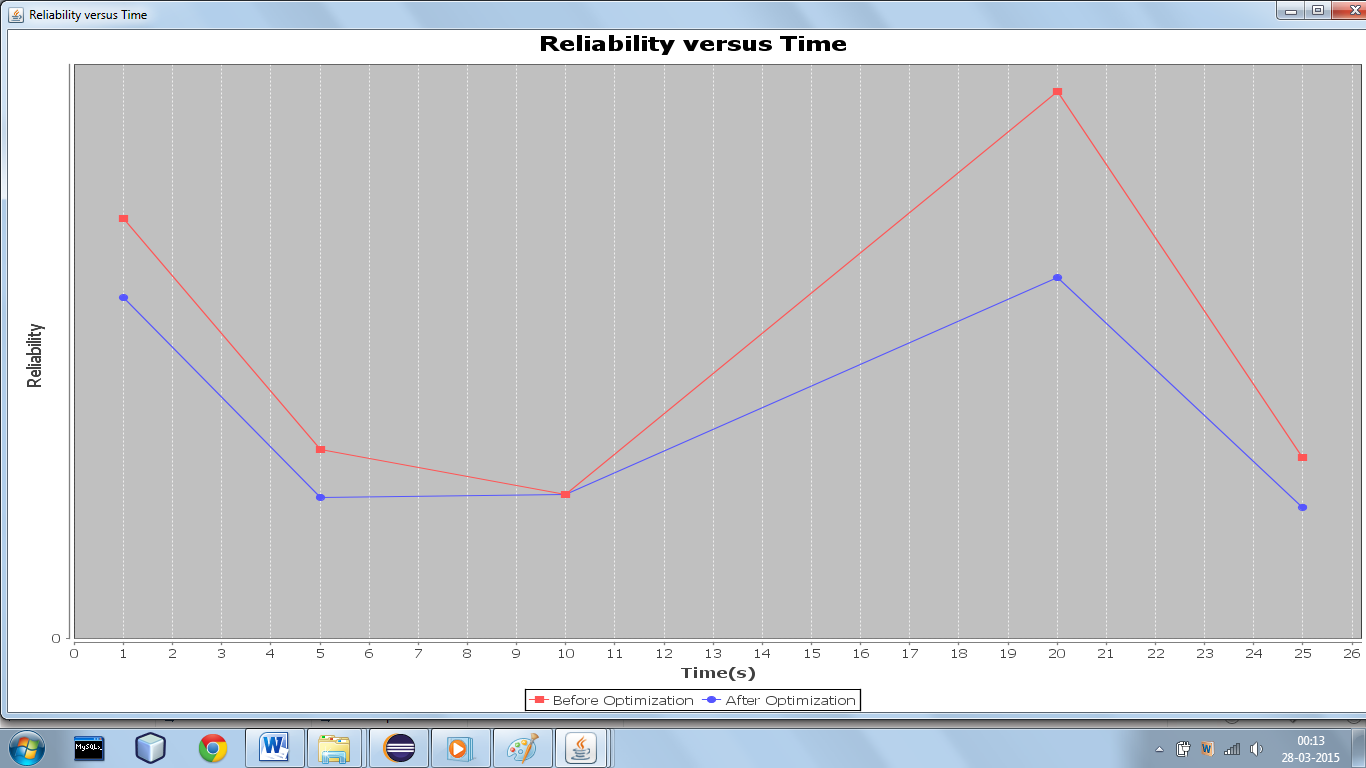
**Failure graph:**

****

*Fig 9 failure graph*

We recorded 21 failures in the first interval, 48 failures in the second interval, 18 intervals in the third interval, 25 intervals in the fourth interval and 32 errors in the final interval, before optimization. The failures were seen to be decreased remarkably after optimization. The number of failures descended to 15 in the first interval,34 in the second interval,13 in the third interval,17 in the fourth interval and 22 in the final interval.

**Reliability Graph:**

****

*Fig 10 Reliability graph*

The growth model represents the reliability or failure rate of a system as a function of time or the number of test cases. The time interval between software failure occurences follows an exponential distribution with mean 1/λn where n=0,1,2,.. denotes the cumulative number of corrected faults.

**CHAPTER 7**

**CONCLUSION**

**CONCLUSION**

With this project it has been shown that the artificial bee colony algorithm works efficiently in optimizing the bank application used. The failures were detected and reduced by the supplication of the optimizing algorithm. The reliability and failures were visually displayed using bar and line graphs. The best time interval during which the maximum number of failures were reduced was noted down.

**CHAPTER 8**

**BIBLIOGRAPHY**

**BIBLIOGRAPHY**

1. D. Karaboga, B. Akay, “*A Comparative Study of Artificial Bee Colony Algorithm, Applied Mathematics and Computation*”, 214, 108-132, 2009.
2. Michael R. Lyu, “*Handbook of Software Reliability Engineering*”, Mc. Graw Hill,April 1996.
3. Hoang Pham, “*System Software Reliability*”, Springer Series in Reliability Engineering,2000.
4. Roger S. Pressman “Software Engineering: A Practitioner's Approach”, Mc. Graw Hill, March 12, 1982.
5. https://en.wikipedia.org.