ISSN (e): 2250 – 3005 || Volume, 07 || Issue, 10|| October – 2017 ||
International Journal of Computational Engineering Research (IJCER)

Soft Computing Approaches for Automated Software Testing

Pradeep Kumar
Associate Professor, Department of CS&IT Maulana Azad National Urdu University, Hyderabad
Corresponding Author: Pradeep Kumar

ABSTRACT
Software testing is most widely used techniques for achieving high quality software. Software testing is done to detect presence of faults, which upon execution causes software failure. Testing the software is time consuming and expensive too. It consumes almost 50% of the software system development resources. Software testing can be defined as process of verifying and validating software to ensure that software meets the technical as well as business requirements as expected. Verification is done to ensure that software meets specification and is close to structural testing whereas validation is close to functional testing done by executing software under test. Testing can be done either manually or automatically by using testing tools. It is empirically observed that automated software testing is better than manual testing. Various techniques have been proposed for generating test data and test cases automatically. Recently, lot of work is being done for test cases generation using soft computing techniques like Fuzzy Logic, Artificial Neural Networks (ANN) and Genetic Algorithm (GA). The main objective of this paper is: (a) study the pertinent issues with respect to automated software testing (b) identify the essential attributes that contributes towards efficiency and quality of automated software testing using Soft Computing Techniques © identify the quality metrics for automated software testing

Keywords: Software Testing, Artificial Neural Network (ANN), Genetic Algorithm (GA), Fuzzy Logic (FL), Software metrics

Date of Submission: 20-10-2017 Date of acceptance: 02-11-2017

I. INTRODUCTION

Software testing is the process of testing the software product contributing to the delivery of high quality software products, lower maintenance costs together with more accurate and reliable results. However, testing is a very expensive process and consumes significant amount of effort. Therefore we require proper strategy to carry out testing activities systematically and effectively. Thus testing strategy provides a framework or set of activities which are essential for the success of the project [1-6]. A more appropriate definition can be described as: “Testing is the process of executing a program with the intent of finding errors”. It is not possible to test the software for all possible combinations of input cases. This way, no software would ever be released if the developers were asked to certify that it was totally free of all errors. Although software testing is quite expensive activity, but then launching of software without testing may lead to cost potentially much higher than that of testing particularly in safety critical systems where human safety is involved. Therefore, testing continues to the extent where it is considered that costs of the testing processes significantly outweigh the returns.

Software testing is indispensable for all software development. It is an integral part of software engineering discipline. However, testing is effort-intensive and expensive activity. It is often accounted for more than 50% of total development costs. Thus, it is imperative to reduce the cost and improve the electiveness of software testing by automating the testing process. In fact, there has been a rapid growth of practices in using automated software testing tools. Currently, a large number of software test automation tools have been developed and become available in the market. Among many testing activities, test case generation is one of the most intellectually demanding tasks and also of the most critical ones, since it can have a strong impact on the effectiveness and efficiency of whole testing process [7-11].

A great amount of research effort in the past decades has been spent on test automation. As a result, several techniques of test case generation, selection, prioritization and optimization has been evolved. On the other hand, software systems have become more and more complicated, for example, with components developed by different vendors, using different techniques in different programming languages and even running on different

platforms. Although automation techniques for test case generation, selection, prioritization and optimization start gradually to be adopted by the IT industry in software testing practice, there still exists a big gap between real software application systems and practical usability of test automation techniques proposed by the researchers.
Therefore for practitioners and researchers it is highly desirable to critically review the existing techniques, recognizing the open problems and putting forward a perspective on the future of test automation with the help of potentially available techniques such as soft computing, machine learning and deep learning.

1.1 Motivation

Software testing is one of the robust and most widely used techniques for achieving high quality software. Software testing is done to detect presence of faults, which cause software failure. However, software testing is a time consuming and expensive task. It consumes almost 50% of the software system development resources. Software testing can also be defined as process of verifying and validating software to ensure that software meets the technical as well as business requirements as expected. Verification is done to ensure that the software meets specification and is close to structural testing whereas validation is close to the functional testing and is done by executing software under test.

Broadly, testing techniques include functional (black box) and structural (white box) testing. Functional testing is based on functional requirements whereas structural testing is done on code itself. Gray box testing is hybrid of white box testing and black box testing. Testing can be done either manually or automatically by using testing tools. It is found that automated software testing is better than manual testing. However, very few test data generation tools are commercially available today. Various techniques have been proposed for generating test data and test cases automatically. Recently, lot of work is being done for test cases generation using soft computing techniques like fuzzy logic, Artificial Neural Networks (ANN) and Genetic Algorithm (GA) [12-17].

1.2 Software Testing

Software testing is one of the robust and most widely used techniques for achieving high quality software. Software testing is done to detect presence of faults, which cause software failure. However, software testing is a time consuming and expensive task. It consumes almost 50% of the software system development resources. Software testing can also be defined as process of verifying and validating software to ensure that software meets the technical as well as business requirements as expected. Verification is done to ensure that the software meets specification and is close to structural testing whereas validation is close to the functional testing and is done by executing software under test.
Broadly, testing techniques include functional (black box) and structural (white box) testing. Functional testing is based on functional requirements whereas structural testing is done on code itself. Gray box testing is hybrid of white box testing and black box testing. Testing can be done either manually or automatically by using testing tools. It is found that automated software testing is better than manual testing. However, very few test data generation tools are commercially available today. Various techniques have been proposed for generating test data and test cases automatically. Recently, lot of work is being done for test cases generation using soft computing techniques like fuzzy logic, Artificial Neural Networks (ANN) and Genetic Algorithm (GA) [12-17].

II. LITERATURE REVIEW

Fraser and Andrea (2013) have shown that optimizing whole test suites toward a coverage criterion is superior to the traditional approach of targeting one coverage goal at a time. The results are significantly better overall coverage with smaller test suites. While they have focused on branch coverage in this paper, the findings also carry over to other test criteria. Consequently, the ability to avoid being misled by infeasible test goals can help in overcoming similar problems in other criteria, for example, the equivalent mutant problem in mutation testing [13-19].
The approach presented in this paper aims at producing small test suites with high coverage such that the developer can add test oracles in terms of assertions. Although keeping the test suites small is helpful in this respect, the oracle problem is still very difficult.
Srivastava and Tai-Hoon, (2009) applied the Genetic Algorithm technique to find the most critical paths for improving software testing efficiency. The authors present a method for optimizing software testing efficiency by identifying the most critical path clusters in a program. This is done by developing variable length Genetic Algorithms that optimize and select the software path clusters, which are weighted in accordance with the criticality of the path. The approach used by the authors is a Weighted Control Flow Graph to test data generation using Genetic Algorithm. Path testing searches the program domain for suitable test cases that cover every possible path in the software under test. Path testing selects a subset of paths to execute and find test data to cover it. The authors conclude that Genetic Algorithm techniques can be applied for finding the most critical

paths for improving software testing efficiency and the Genetic Algorithms also outperform the exhaustive search and local search techniques. The experiments conducted by the authors were based on small examples.

The concept of dominance relations between the nodes of Control Flow Graph to reduce the software testing cost has been proposed by Ghiduk and Girgis, (2010). A new fitness function is defined using dominance relationship to evaluate the generated test data. Experiments have been carried out to evaluate the effectiveness of the proposed GA technique as compared to the Random Testing (RT) technique, and to evaluate the effectiveness of the new fitness function and the technique used to reduce the cost of software testing. The results showed that the proposed GA technique outperformed the RT technique. The proposed GA and RT techniques were applied to object oriented C++ programs. Testing on other type of programs, like structure oriented program, should be done for wide acceptability of the results.

Yong, (2009) presented an approach of generating test data for a specific single path based on genetic algorithms. A similarity between the target path and execution path with sub path overlapped is taken as the fitness value to evaluate the individuals of a population and drive GA to search the appropriate solutions. The authors conducted several experiments to examine the effectiveness of the designed fitness function, and evaluated the performance of the function with regards to its convergence ability and consumed time. Results show that the function performs better as compared with the other two typical fitness functions for the specific paths employed by the authors.

Michael, (1997) proposed that the combinatorial optimization techniques when used with Genetic algorithms solve problems involving the simultaneous satisfaction of many constraints like, condition based decision coverage. The authors have performed experiments for comparing random test data generation with GA by applying genetic search. The genetic search outperformed random test generation, but the authors' experiment did not include the implementation of path selection heuristic for the generation of test data for programs with procedures.

Nirpal and Kale, (2010) have compared the software test data for automatic path coverage using genetic algorithm with Yong Chen approach (Chen Yong, 2009) of generating test data for path testing. They proved that the genetic algorithm approach outperforms the Yong Chen approach. The genetic algorithm is found useful in reducing the time required for lengthy testing by generating the meaningful test cases for path testing. The genetic algorithm is required to be build for structural testing for reduce execution time by generating more suitable test cases.

Faezeh (2005) focused on the use of independent path to reduce time and on precisely monitoring the execution trace of the program. Genetic algorithm is applied with improved parameters for test cases designed to better detect bugs of tested program. The genetic algorithm based tester fulfills test criteria by manner of evolutionary computation. Genetic Algorithm method with dynamic fitness function and stopping criterion is used for effective testing and low cost identification of infeasible path. The approach used suffers from the disadvantage about dynamic aspect of testing, as the stopping criteria used can't specify actual number of generations, i.e. in some cases, the tester is exited based on waiting time, while the stopping criterion is not satisfied.

Last Mark., (2005) stated that Fuzzy-based Age Extension of Genetic Algorithm is more efficient to generate black box test cases than Simple Genetic Algorithm because the probability of finding the error with the former approach is much more as compared to the latter, which results in saving a lot of resources for the testing team. The number of distinct solutions produced is significantly higher, which is useful for investigation and identification of the error itself by the software programmers. There is need to apply the proposed methodology to test real programs and to develop evaluation functions for the evolved test cases.

Genetic Algorithm approach has been used by Gupta and Rohil, (2008) to generate test cases for Object Oriented Software where statements are represented in the form of a tree. The test cases are encoded using various strategies and objective functions. Test cases for testing object oriented software include test programs which create and manipulate objects in order to achieve a certain test goal. The approach described by them facilitates the automatic generation of object oriented test program using genetic algorithms. The approach has been used to generate the test cases for Java classes to prove the concept.

Rajappa, (2008) proposed graph theory based genetic approach to generate test cases for software testing. In this approach the directed graph of all the intermediate states of the system for the expected behavior is created. The base population of genetic algorithm is generated by creating a population of all the nodes of the graph. A pair of nodes referred to as parents are then selected from the population to perform crossover and mutation on them to obtain the optimum child nodes. The process is continued until all the nodes are covered. This process is followed for the generation of test case in the real time system. The technique is more concrete in case of network testing or any other system testing where the predictive model based tests are not optimized to produce the output.

As proposed by Berndt, (2003) Genetic algorithm is used to breed software test cases. The Genetic algorithm includes a fossil record that records past organism, allowing any current fitness calculation to be influenced by the past generations. Factors developed for fitness functions are novelty, proximity and severity. Novelty is a

measure of the uniqueness of a particular test case, Proximity is the measure of closeness to other test cases that resulted in system failures and Severity is the measure of seriousness of system error. Interplay of these factors produces complex search behavior.
Rauf, (2010) have presented a Graphical User Interface (GUI) testing and coverage analysis technique entered on genetic algorithms in order to exploit the event driven nature of GUI. Event-flow graph technique is used in the field of automated GUI testing. Just as the control-flow graph, GUI that represents all possible execution paths in a program, event-flow model, in the same way, represents all promising progressions of events that can be executed on the GUI. Genetic algorithm searches for the best possible test parameter combinations according to predefined test criterion, like coverage function that measures how much of the automatically generated optimization parameters satisfy the given test criterion.

III. TESTING METHODOLOGY AND TOOLS

As the software industry grows, it becomes more and more competitive and advanced for businesses to produce good quality software. The production of the software and the quality together must be increased for businesses to produce the best possible software. Testing, therefore, has to be done throughout the process of programming the software. Manual testing takes too long and can consume a lot of time. With the aid of testing tools this can increase efficiency and get the deadlines met. There are different methodologies that can be used when implementing testing tools. The testing methodologies can be used together or individually. It all depends on what you are testing as they provide better resources for different tasks. Below are commonly used testing tools described as below [7-10]:

S.No.	**Methodology**	**Description**	**Commonly used Testing Tool**
1	Unit Testing	Unit testing is the process of taking a module and executing it in isolation from the remaining software modules by prepared test cases and comparing the actual results with the predicted results through specification and design of the module. The purpose of unit testing is to determine whether each independent module is correctly implemented or not. In context of procedural programming, the unit can be any individual function or a procedure which is written in the same language as the production code to be tested.	JUnit Nested Runner Junit dataprovider JUnit is a very popular testing tool for unit testing
2	Integration Testing	The integration testing is performed to ensure that interface between modules is working correctly. Main purpose of integrating testing is the interface to check whether the parameters match both sides as to type, their permissible range, meaning and utilization of the independent modules. There are three main integration strategies to interface between the modules such as top-down approach, bottom-up approach and sandwich approach. Top-down integration proceeds down the invocation hierarchy adding one module at a time until the entire tree level. Bottom-up approach works from the bottom to the top in a tree level structure. Sandwich is the hybrid category and starts from top and bottom concurrently and meeting somewhere in the middle.	Quick Test Professional (QTP) is an automated functional Graphic User Interface (GUI). QTP can test many applications such as Java, visual basic application, .net and many more HPE Unified Functional Testing (HP- UFT)
3	System Testing	During system testing, we need to evaluate a number of attributes of the software that are vital to the user such as security, compatibility and dependentability. Testing the system's capabilities is more important than testing its components. Therefore we should focused on finding failures those are catastrophic rather than the failures that are merely annoying.	Functional Testing Tools such as Salenium, Watir, WatIN, WET, SAMIE, WebInject are commonly used automated tools
4	Validation Testing	Validation testing is performed after unit and integration testing is over. The software is tested as a complete product where we want to test the software with the perspective of the customer and would like to ensure that the software meets the expectations of the customers. We also check all the entries of validation criteria described in software requirement specifications (SRS) of SDLC. Further in order to improve the confidence level and familiarity with the software, the involvement of customers is required during validation testing. Some well-known and commonly used methods for maximizing the customer's involvement are alpha, beta and acceptance testing, the method of validation techniques to involve the customers during testing.	In addition to this, the IEEE has developed a standard (IEEE std. 1059-1993) entitled "IEEE guide for software verification and validation" to provide specific guidance about planning and documenting the tasks required by the standard so that the customer may write an effective plan [IEEE93].
5	White Box Testing	Also known as clear box testing, glass box testing, translucent box testing or structural testing. It uses the internal perspective of the system and then designs test cases based on this internal structure. Using this method, the code itself and all the conditions, statements and paths along with it are tested. Programming skills are required for	Code Review or Walkthrough Path Analysis Code coverage analysis

		noticing all the paths through the software. White box testing does many things such as analyzing the data flow, control flow, information flow and coding practices. Testing plans are made according to the details of the software implementation, such as programming language, logic, and styles. Test cases are derived from the program structure.	
6	Black Box Testing	The black-box approach is a testing method in which test data are derived from the specified functional requirements without regard to the final program structure. It is also termed data-driven, input/output driven, or requirements-based testing. Because only the functionality of the software module is of concern, black-box testing also mainly refers to functional testing. Testing method emphasized on executing the functions and examination of their input and output data only. The tester treats the software under test as a black box, only the inputs, outputs and specification are visible, and the functionality is determined by observing the outputs to corresponding inputs. In testing, various inputs are exercised and the outputs are compared against specification to validate the correctness.	Webdriver Appinum Testing Applitools Eyes Testing of Mobile apps, web and desktop apps HP QTP
7	Regression Testing	Testing the application as a whole for the modification in any module or functionality. Difficult to cover all the system in regression testing so typically automation tools are used for these testing types. The regression testing may take place once the programmer has tried to fix a problem or has purposely added in code to give out errors.	GUI: Auto Testers for Windows OTF – Object Testing Framework QTP HP-UFT
8	Web based Testing	This type of testing is used over web based applications. It tests for bugs and problems that appear within the application. An example of this may be that once a problem is detected such as having dead links, line checking or html validation then it provides help in repairing this problem.	Selenium LoadTracer tool is used for web servers and to analyze the performance and characteristics of the web application. LoadTracer is a GUI tool and checks the loading and performance and scalability of the web application.
9	Security Testing	Security of the system testing can help against unauthorized access, hacking and any coding damage which deals with the code of application. Software quality, reliability and security are tightly coupled. Flaws in software can be exploited by intruders to open security holes. Many critical software applications and services have integrated security measures against malicious attacks. The purpose of security testing of these systems include identifying and removing software flaws that may potentially lead to security violations, and validating the effectiveness of security measures.	IBM Internet security scanner HP WebInspect DevInspect SPIKE Metasploit Wireshark Back Track Retina and Quality Assurance (QA) inspect
10	Functional Testing	This type of testing ignores the internal parts and focus on the output is as per requirement or not. Black-box type testing geared to functional requirements of an application.	Silk Test Test Complete Quick test Pro Rational Robo, SoapTest RFT, BadBoy
11	Performance Testing	It is similar to using web tools but has its own applications. An example program is LoadTracer which is a GUI tool and checks the loading and performance and scalability of the web application.	LoadTracer SilkRealizer Cloud Test Load Storm
12	Database Testing	Database testing tools are very good to use for checking and testing databases. It helps with creating tables and data to test the database.	SQL DB Validator performs database and data cube verification and validation.
13	Static Testing Tool	It checks the systems software and does not execute the program itself	Lint, Testbed, ParaSoft VERACODE CAST
14	Communication Testing	Communications testing tools are used for communication purposes. It can work wirelessly, through sockets and SOAP, GPRS and other network communication methods.	Cheetah - It allows you to proactively test and monitor your VoIP and VoD performance whilst it maintains the integrity of other critical applications.
15	Requirement Management	This is used to analyze the requirements for testing, maintaining and logical inconsistencies. This helps the tester to validate the program correctly. Requisite Pro is a management tool to help improve communication goals and enhance collaborative development and increase the quality of applications before deploying.	IBM.s Requisite Pro
16	Performance Testing	Performance evaluation of a software system usually includes: resource usage, throughput, stimulus-response time and queue lengths detailing the average or maximum number of tasks waiting to be serviced by selected resources. Typical resources that need to be	The typical method of doing performance testing is using a benchmark, workload or trace.

		considered include network bandwidth requirements, CPU cycles, disk space, disk access operations, and memory usage. The goal of performance testing can be performance bottleneck identification, performance comparison and evaluation, etc.	Jmeter FunkLoad
17	Reliability Testing	Software reliability refers to the probability of failure-free operation of a system for a specified period of time in a specified environment. Testing is an effective sampling method to measure software reliability. Guided by the operational profile, software testing (usually black-box testing) can be used to obtain failure data, and an estimation model can be further used to analyze the data to estimate the present reliability and predict future reliability.	SoRel, CASRE, SARA, ESTM, RGA ReliaSoft Based on the estimation, the developers can decide whether to release the software, and the users can decide whether to adopt and use the software. Risk of using software can also be assessed based on reliability information
18	Acceptance Testing	Perform subset of system test as part of demonstration of user acceptance test. Normally this type of testing is done to verify if system meets the customer specified requirements.	Load performance & Simulation Tools: User or customer do this testing to determine whether to accept application or not
19	End-to-End Testing (E2E)	It is similar to system testing, involves testing of a complete application environment in a situation that mimics real-world use, such as interacting with a database, using network communications, or interacting with other hardware, applications, or systems.	White Box testing or Black box testing can be applied for E2E testing depending upon the situational context
20	Load Testing	Its performance testing to check system behavior under load. Testing an application under heavy loads, such as testing of a web site under a range of loads to determine at what point the system's response time degrades or fails.	Web Load Professional HP Load Runner Load Tracer Forecast VPerformer
21	Stress Testing	System is stressed beyond its specifications to check how and when it fails. Performed under heavy load like putting large number beyond storage capacity, complex database queries, continuous input to system or database load	Web Load LoadView Apache Jmeter HP Load Runner
22	Usability Testing	User-friendliness check. Application flow is tested, Can new user understand the application easily, Proper help documented whenever user stuck at any point. Basically system navigation is checked in this testing.	Optimizely, Crazy Egg TryMyUI, Qualaroo, Usabilla, Feedback Army UserFeel

IV. SOFT COMPUTING TECHNIQUES

Soft Computing methodologies are designed to model and enable solutions to real world problems, which are not modelled or too difficult to model mathematically. Soft Computing is basically optimization technique to find solution of problems which are very hard to answer. Soft computing is a consortium of methodologies that provides flexible information processing capability for handling real-life ambiguous situations [9-11].

Soft Computing aims to exploit the tolerance for imprecision, uncertainty, approximate reasoning and partial truth in order to achieve tractability, robustness and low-cost solutions. The guiding principle behind soft computing is to devise methods of computation that lead to an acceptable solution at low cost, by seeking for an approximate solution to an imprecisely or precisely formulated problem

4.1 Fuzzy Logic

Fuzzy Logic process the data by allowing partial set membership rather than crisp set membership or non-membership. Fuzzy Expert System consists of fuzzification unit that converts crisp values into fuzzified input. It consists of inference engine that contains if then else rules and a defuzzification unit to convert the result in a readable form. Fuzzy Logic incorporates a simple, rule-based IF X AND Y THEN Z approach to a solving problem rather than attempting to model a system mathematically.

4.2 Neural Network

A Neural network, more properly referred to as an 'artificial' neural network (ANN), is a computing system made up of a number of simple, highly interconnected processing elements, which process information by their dynamic state response to external inputs. The representation of knowledge is distributed over these connections and "learning" is performed by changing certain values associated with such connections, not by programming. The ability of Artificial Neural Networks (ANNs) to model the complex non-linear relationships and capability of approximating any measurable function make them attractive prospects for solving complex tasks without having to build an explicit model of the system.

A neural network is the way of learning mechanism designed in a similar fashion in which the brain performs a given task. It consists of large number of simple processing elements called neurons connected to each other by direct communication links associated with weight. The neural network is trained through supervised learning method by providing network with a series of sample inputs and comparing the expected sample output with responses over a prespecified period of time. The training procedure carried out until network is able to provide expected and convincing responses. The neurons are arranged into layers and the patterns of connection within and in-between layers constitute the architecture of network. The number of layers in the network can be either single or multiple based on the number of layers of weighted interconnected links between the particular slabs of neurons.

4.2.1 Artificial Neural Network Modeling

A neural network is the learning mechanism, which can be utilized for reliability prediction in similar fashion in which the brain performs a given task. It consists of large number of simple processing elements called neurons connected to each other by direct communications/links associated with some weight. The neural network can be trained through supervised or unsupervised learning method by providing network with a series of sample input and comparing the expected sample output with the responses over a specified period of time. The training procedure carried out until network is able to provide expected responses. The neurons are arranged into layers and the patterns of connection within and in-between layers constitute the architecture of network.

The number of layers in the network can be either single or multiple based on the number of layers of weighted and interconnected links between the particular slabs of neurons [9]. Mathematically, the single layer neural network model can be defined as:

$$net_k = b_k + \sum_{i=1}^{n} x_i w_{ki} \text{ and } y_k = f(net_k) \qquad (1)$$

where n is number of input elements i.e., $x_1, x_2, x_3, \ldots\ldots\ldots\ldots x_n$

f() = activation function used for processing the input signals and generating the final output of the neuron; ∑ = summation function

w_{kj} =a set of connecting links associated with weighs $w_{k1}, w_{k2}, w_{k3}, w_{k4}, \ldots\ldots\ldots\ldots w_{kn}$.

Y_k = output of the previous layer of network

Thus, ANN is an information processing system composed of neurons, which can be applied for mapping past failure behavior of software to predict future failure trends shown as in Figure1.

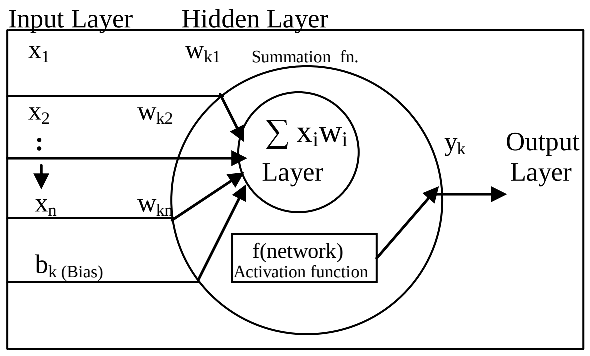

Figure1. Processing of the neuron in feed forward multilayer network

The input signal of neural network is transmitted by using connection links associated with weight, which is multiplied with incoming signal for the neural networks The output signal is obtained by applying activation function to the input signal. The neural network can be either single layer or multilayer networks.

4.2.2 System Architecture

The system architecture of proposed neural network model comprised of three component neural network where each component neural network is three-layer single-input single-output feed forward neural network (FFNN) containing N_h nodes in the hidden layer. The proposed neural networks are trained using supervised learning algorithms by adjusting different weights connecting these layers. The output of component neural network is combined all together to produce final output of the model.

4.2.3 Components of ANN Architecture

The architecture of proposed model using back propagation algorithm is shown in figure2. The sigmoidal function f(x) is taken as an activation function, which can be written as:

$$f(x) = \frac{1}{\left(1 + e^{-\sigma x}\right)} \qquad (2)$$

where σ is the learning rate used for the adjustment of weights.

There are millions of simple processing elements or neurons in the brain, linked together in a massively parallel manner. This is believed to be responsible for the human intelligence and discriminating power. Neural Networks are developed to achieve biological system type performance using a dense interconnection of simple processing elements analogous to biological neurons.

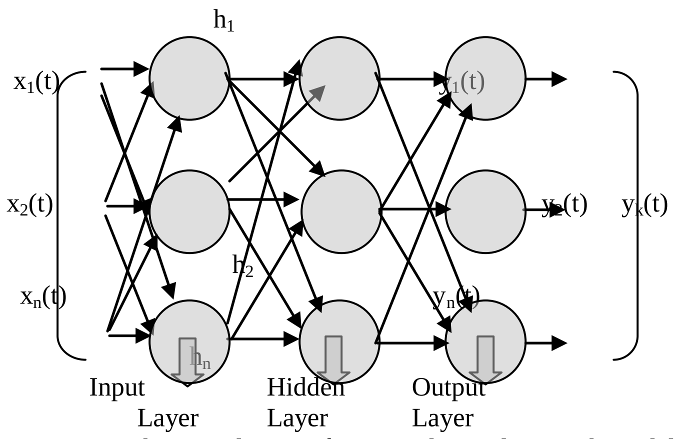

Figure2. Architectural view of proposed neural network model

Neural Networks are information driven rather than data driven. Typically, there are at least two layers, an input layer and an output layer. One of the most common networks is the Back Propagation Network (BPN) which consists of an input layer, and an output layer with one or more intermediate hidden layers. The major issues of concern today are the scalability problem, testing, verification and integration of neural network systems into the modern environment. Neural network programs sometimes become unstable when applied to larger problems. Also there are some more practical problems like: the operational problem encountered when attempting to simulate the parallelism of neural networks. Since the majority of neural networks are simulated on sequential machines, giving rise to a very rapid increase in processing time requirements as size of the problem expands. Networks function as "Black Boxes" whose rules of operation are completely unknown [9].

4.2 Genetic Algorithms

GAs is general-purpose search algorithms, which uses principle inspired by natural genetics to evolve solutions to problems [25-28]. GA starts off with population of randomly generated chromosomes, each representing a candidate solution to the concrete problem by applying genetic operators based on the genetic processes occurring in nature. GAs had a great measure of success in search and optimization problems due to their robust ability to exploit the information accumulated about an initially unknown search space. Particularly GAs specialize in large, complex and poorly understood search spaces where classic tools are inappropriate, inefficient or time consuming.

GA's basic idea is to maintain a population of chromosomes. This population evolves over time through a successive iteration process of competition and controlled variation. Each state of population is called generation. Associated with each chromosome at every generation is a fitness value, which indicates the quality of the solution, represented by the chromosome values. Based upon these fitness values, the selection of the chromosomes, which form the new generation, takes place. Like in nature, the new chromosomes are created using genetic operators such as crossover and mutation.

4.2.1 Mechanism of GA

The fundamental mechanism consists of the following stages:

a. Generate the initial population randomly
b. Select the chromosomes with the best fitness values
c. Recombine selected chromosomes using crossover and mutation operators
d. Insert offspring into the population
e. If stop criterion is satisfied, return to the chromosome(s) with the best fitness. Otherwise, go to 2nd step (b)

In GA, the population is defined to be the collection of individuals. A population is a generation that undergoes under changes to produce new generation. Like nature, GAs have also collection of several members to make population healthy. A chromosome that is a collection of genes is correspondence to individual of population. Each individual chromosome represents a possible solution to the optimization problem. The dimension of the GA refers to the dimension of the search space which equals the number of genes in each chromosome.

4.2.2 Representation of Chromosomes

The representation of chromosomes in GAs has very deep impact on the performance of GA-based function. There are different methods of representation of chromosomes like binary encoding, value encoding, permutation encoding, tree encoding etc. The most commonly used encoding is binary encoding proposed by Holland. In this method, the value of individual is encoded as bit string consists of binary values either 0 or 1. Each chromosome of population consists of same length of binary string [21-27].

V. MEASUREMENT TECHNIQUES

Two well-known and widely used reliability metrics are categorized according to the number of failures in a time period and time between failures. Another important metric, fault and failure metrics describe the product reliability characteristics such as mean time between failures (MTBF) and failure density (FD).The main goal of evaluating fault and failure metrics is to determine when the software is attaining failure-free execution [9], [10].

The number of faults found during testing and failures reported by users after delivery are collected, summarized and analyzed in order to achieve better reliable software. Thus, testing strategy is highly relative to the effectiveness of fault metrics, since if the testing scenario does not cover full functionality of the software, then it may pass all tests and yet be prone to failure after delivery. Usually, failure metrics are based upon customer information regarding failures found after release of the software. The failure data collected is therefore used to calculate MTBF and FD to measure the software reliability. The current practices of software reliability measurement can be divided into three broad categories such as product metrics, process metrics and project metrics described as follows:

5.1 Product metrics

Product metrics describe the characteristics of software product in terms of size, complexity, design features, performance, efficiency, reliability, portability, etc.

5.2 Process metrics

Process metrics describe the effectiveness and quality of the processes that produce the software products. Example of process metrics includes such as effort estimation required in the process, time to produce the product, effectiveness of defect removal during development, number of defects found during testing, maturity of the process.

5.3 Project metrics

Project metrics describe the project characteristics and execution in terms of several parameters including number of software developers, staffing pattern over the life cycle of the software, cost, schedule and productivity. Software reliability metrics can help to improve software engineering processes through quantitative evaluation of software technologies, tracking software development status and conducting maintenance activities.

5.4 Useful Metrics for Software Reliability

In this section we are focused on intrinsic product quality metrics and customer satisfaction metrics. Intrinsic product quality is measured using number of faults in the software or by estimating that how long the software can run before encountering a failure. Some useful metrics applied for test automation are described as follows:

5.4.1 Mean Time to Failure (MTTF)

The MTTF metric measures the time between failures (TBF) and usually applied in various critical safety systems such as ATC (air traffic control system) and space shuttle control systems. Thus MTTF metric is more appropriate for special-purpose and safety critical software systems.

Mean Time to Failures (MTTF): $MTTF = \frac{1}{\lambda}$

Where λ, is the failure intensity function.

5.4.2 Failure Rate (FR)

The failure rate or failure intensity is defined as the number of failures per unit time. Thus, fault and failure metrics describes the product reliability characteristics such as Mean time between failures (MTBF) and Failure density (FD).

$$FR = \frac{\text{Number of Failures}}{\text{Execution time}}$$

5.4.3 Defect Density Rate (DDR)

Defect density rate metric measures the defects relative to the software size using Source Lines of Codes (SLOC) and Function Points (FPs). Thus defect density is a measure of the total known defects found in software divided by the size of software being measured. The total number of known defects is the count of defects identified against particular software during a particular time period. The time period may be represented in terms of no. of hours/days/weeks/months/year.

$$DDR = \frac{\text{Total number of known defects}}{\text{Size}}$$

DDR is most often used in many commercial software systems such as billing systems, online reservation system, office automation and other MIS (management information systems). DDR can be used to compare the relative number of defects in various software components or different layers of the proposed model in this work. Another important use of DDR is to compare subsequent releases of a product in order to track the impact of defect reduction and quality improvement activities.

5.4.4 Total Released Defects (TRD)

$$TRD = \frac{\text{Number of released defects}}{\text{Total SLOC}}$$

Thus, identifying defect prone software or components allows the concentration of limited resources in the areas with highest potential return on the investment for any commercial applications.

5.4.5 Failure Intensity Improvement Factor (FIIF)

The reliability growth as a result of testing such as failure intensity improvement factor is defined as follows:

$$FIIF = \frac{\text{Value of failure intensity at the start of testing}}{\text{Value of failure intensity at the end of testing}}$$

5.4.6 Reliability Growth Factor (RGF)

To find the growth of reliability from proposed model's (software reliability growth model for three tier client server system) viewpoint, the reliability growth factor is defined as follows:

$$RGF = \frac{\text{Final time between failures}}{\text{Initial time between failures}}$$

for

$$PassedTestcasesPercentage = \left(\frac{NumberofpassedTests}{Tota \ln umberoftestexecuted}\right) \times 100$$

TBF models

$$RGF = \frac{\text{Initial failure rate}}{\text{Final failure rate}}$$, for FC fault count model

5.4.7 Customer Oriented Metrics

The customer oriented metric is another product quality metric used by developers and industry professionals to measure the problems encountered by the customers during the operational phase. The problem metric can be expressed in terms of problems per user month (PUM) computed as follows:

$$PUM = \frac{\text{Total no. of problems reported by the customers for a time period}}{\text{Total number of license} - \text{months of the software during the period}}$$

where, Number of license-months = (Number of install licenses of the software) * (Number of months in the calculation period)

PUM is calculated for each month after the software is released to the end customer and we intend to achieve low value of PUM.

The customer problems metric can be regarded as an intermediate measurement between defects measurement and customer satisfaction. However, to reduce the customer problems we need to reduce the functional defects in the products together with other attributes such as usability, documentation, problem rediscovery etc.

5.4.8 Customer satisfaction

Customer satisfaction can be measured by using customer survey on five-point scale such as customer is highly satisfied, customer is satisfied, neutral, unsatisfied and strongly unsatisfied. Further, based on five-point scale, several other metrics are designed as follows: (customer satisfaction index CSI on the scale of 0 to 10)

a. Percentage of highly satisfied customers are categorized as very happy customers (above 7 point scale)
b. Percentage of satisfied customers are categorized as happy customers (between 5 to 7)
c. Percentage of not satisfied customers are categorized as unhappy customers (between 3 to 5 point scale)
d. Percentage of neutral customers are categorized as normal customers who is neither happy nor unhappy but they are utilizing and continuing with software systems (between 2 to 3 point scale)
e. Strongly unsatisfied customers are categorized as very unhappy customers (below 2 point scale)

In practice we tend to minimize the percentage of not satisfied customers together with neutral and completely unsatisfied customers.

5.4.9 Failure Behavior metrics

Failure behavior of software generally depends on the environment and the number of faults present in the program during execution. In classical reliability theory, failure occurrences are expressed as random variables due to the unpredictable nature of fault introduction by the programmers, and unpredictable conditions under which programs are executed. There are four types of metrics commonly used to represent failure occurrences using time variable such as time of failures, time interval between failures, cumulative failures occurred up to a specified time and failures occurred in a specified time interval.

5.4.10 Reliability metrics

Another very important and useful metrics are software reliability metrics which are derived from the failure occurrence and failure datasets. Some commonly used well-known reliability metrics are summarized in Table1 [1],[6],[9],[10].

Table1. Software reliability metrics

Metrics	Description
Absolute metrics	Fundamental metrics for software testing includes: Total number of test cases generated, Number of test cases passed / Failed, Number of test cases blocked, number of defects found/accepted/rejected/deferred, number of defects found after release of the product etc.
POFOD	Probability of Failure on Demand (POFOD) is the metric applied to any software systems where critical service request are happened in an unpredictable way, or when there is a long time interval between consecutive requests.
ROCOF	Rate of Occurrence of Failures (ROCOF) is a robust metric used in software systems where (critical) services are demanded in more regular way.
MTTF	Mean Time To Failure (MTTF) is very common metric employed to software systems involving long transactions, during which a guarantee of service continuity and delivery is expected. MTTF is the average time it takes for a system to fail.
AVAIL	Availability is the likelihood that the system will be working at a given time. That is, AVAIL is a metric for software systems where continuous service delivery is a major concern.
MTTR	MTTR (Mean Time To Recover) is the average time for the system to recover; correspond to the average time to repair the system.
MTBF	MTBF (Mean Time Between Failure) is the average time between consecutive system failures. MTBF is equal to the sum of the MTTF and the MTTR. That is, MTBF = MTTF + MTTR

Table2. Testing Metrics for Monitoring Progress & Quality Assurance

SNo.	Testing Tracking Metrics
1	$Passed\ Test\ Cases\ Percentage = \left(\frac{Number\ of\ passed\ Tests}{Total\ Number\ of\ test\ executed}\right)\times 100$
2	$Failed\ Test\ Cases\ Percentage = \left(\frac{Number\ of\ failed\ Tests}{Total\ Number\ of\ test\ executed}\right)\times 100$
3	$Blocked\ Test\ Cases\ Percentage = \left(\frac{Number\ of\ Blocked\ Tests}{Total\ Number\ of\ test\ executed}\right)\times 100$
4	$Fixed\ Defects\ Percentage = \left(\frac{Defects\ Fixed}{Defects\ reported}\right)\times 100$
5	$Defects\ Deferred\ Percentage = \left(\frac{Defects\ deferred\ for\ future\ releases}{Total\ Defects\ reported}\right)\times 100$
6	$Critical\ Defects\ Percentage = \left(\frac{Critical\ Defects}{Total\ Defects\ reported}\right)\times 100$
7	$Number\ of\ tests\ run\ per\ time\ period = \left(\frac{\text{Number of tests run}}{\text{Total time}}\right)$

8	$Test\ design\ efficiency = \left(\frac{\text{Number of tests designed}}{\text{Total time}}\right)$
9	$Test\ review\ efficiency = \left(\frac{\text{Number of tests reviewed}}{\text{Total time}}\right)$
10	$Number\ of\ defects\ per\ test\ hour = \left(\frac{\text{Total number of defects}}{\text{Total number of test hours}}\right)$
11	$Number\ of\ bugs\ per\ test = \left(\frac{\text{Total number of defects}}{\text{Total number of tests}}\right)$
12	$Number\ of\ bugs\ per\ test = \left(\frac{\text{Total number of defects}}{\text{Total number of tests}}\right)$
13	$Test\ Execution\ Coverage\ Percentage = \left(\frac{Nuumber\ of\ tests\ run}{Total\ number\ of\ tests\ to\ be\ run}\right) \times 100$
14	$Requirements\ Coverage\ Percentage = \left(\frac{Nuumber\ of\ requirements\ Covered}{Total\ number\ of\ requirements}\right) \times 100$

VI. IMPACT OF PROPOSED STUDY ON ACADEMICS/INDUSTRY

The impact of presented study in this paper on academics & industry may be described as follows:

a. Reduction in cost and time in automated software testing will lead to the timely delivery of the product that will boost customer satisfaction up to the larger possible extent
b. Through automation of test cases, it will help the industry professionals in improving the quality aspect of the software product quantitatively as per local needs and global standards.
c. With the help of proposed study on Automated Software Testing using Soft Computing Techniques, the results and performance of automated testing can be improved significantly meeting customer's requirement and quality parameters within time and budget constraints leading to the higher confidence among researchers/practitioners and customer satisfaction

VII. SUMMARY

Software testing is resource consuming and costly too. Automation is a good way to cut down time and cost. In order to automate the process, we need to have some ways to generate oracles from the specification, and generate test cases to test the target software against the oracles to decide their correctness. In general, significant amount of human intervention is also needed in testing because the degree of automation remains at the automated test script level written by testing team. Testing is potentially endless so we cannot test till all the defects are unearthed and removed. It is not economically viable solution. Therefore at some point, we have to stop testing and release the software. Testing is a trade-off between budget, time and quality. The optimistic stopping rule is to stop testing when either reliability meets the requirement, or the benefit from continuing testing cannot justify the testing cost. This will usually require the use of software reliability growth models to evaluate and predict reliability of the software under test.

Using testing to locate and correct software defects can be an endless process. Bugs cannot be completely ruled out. Sometimes testing and fixing problems may not necessarily lead to improve the quality and reliability of the software. Because fixing a problem may introduce other severe problems into the system. Although it can reduce costs by using software testing tools however it may not always be the case. Another problem with the tools is that with using the testing tools there has to be enough knowledge and resources in able to use them.

In order to analyse the performance of various software metrics discussed in this paper, we require basically three types of data sets such as: training dataset, validation dataset and testing dataset. Training data set is required for parameter optimization. Validation data set is used in order to control the bias-variance trade-off, and validation data is used to select the best parameter setting. The testing dataset required for the performance evaluation of reliability prediction method is carried out on unknown (previously unseen) data that has not been used previously to determine the parameters of the prediction method.

REFERENCES

[1]. Aggarwal K.K. and Singh Y. (2012). Software Engineering: New Age International Publishers, India: third edition, pp: 308-346.
[2]. Software Quality Assurance: from theory to implementation, Daniel Galin, Pearson Education, 2004.
[3]. John H. (1999), International trends in software engineering and quality system standards: Ontario Hydro's Perspective, Part 1. Software Quality Professional, 1(2), pp: 51-58.
[4]. Jung-Hua L. (2009), The Implementation of artificial neural networks applying to software reliability modeling. In Proceedings of the Conference on Chinese Control and Decision Conference (CCDC 2009), 17-19 June, Taiwan, pp: 4349-4354.
[5]. G. Bernet, L. Bouaziz, and P. LeGall, "A Theory of Probabilistic Functional Testing," Proceedings of the 1997 International Conference on Software Engineering, 1997, pp. 216 –226
[6]. B. Beizer, "Software Testing Techniques," Second Edition, Van Nostrand Reinhold Company Limited,1990, ISBN 0-442-20672-0
[7]. Software Testing Techniques, Technology Maturation and Research Strategies, Paul Li and Lu Luo School of Computer Science, Carnegie Mellon University
[8]. Automated Software Testing, Elfriede Dustin, et all, Addison Wesley Longman, Inc. July 1999 and Linz, T., Daigl, M. GUI Testing Made Painless.
[9]. Haykin S. (2010). Neural Networks and Learning Machines, third edition PHI, 2010.
[10]. Stephen H.K. (2003). Metrics and models in software quality engineering. Pearson education, India.
[11]. Aggarwal KK, Singh Y, Kaur A, Malhotra R (2006) Investigating the effect of coupling metrics on fault proneness in object-oriented systems. Software Quality Professional 8(4): 4-16
[12]. Aggarwal KK, Singh Y, Kaur A, Malhotra R (2009) Empirical analysis for investigating the effect of object-oriented metrics on fault proneness: a replicated case study. Software Process Improvement Practice 14(1): 39–62
[13]. Malhotra R, Kaur A, Singh Y (2011) Empirical validation of object-oriented metrics for predicting fault proneness at different severity levels using support vector machines. Int J Syst Assur Eng Manag (July-Sept 2010) 1(3): 269–281. DOI 10.1007/s13198-011-0048-7
[14]. Malhotra R, Singh Y, Kaur A (2009) Comparative analysis of regression and machine learning methods for predicting fault proneness models. International Journal of Computer Applications in Technology 35(2): 183-193
[15]. Pradeep Kumar and Yogesh Singh (2010) "Prediction of Software Reliability Using Feed Forward Neural Network", published in the proceedings of International Conference on Computational Intelligence and Software Engineering (CiSE2010), Wuhan, China, ISBN: 978-1-4244-5391-7, DOI 10.1109/CISE.2010.5677251.
[16]. Pradeep Kumar and Yogesh Singh (2012) "An empirical study of software reliability prediction using machine learning techniques" published online in International Journal of System Assurance Engineering and Management (IJSAEM), Vol. 3, No.3, pp: 194-208, Springer Publications. DOI: 10.1007/s13198-012-0123-8.
[17]. Pradeep Kumar and Yogesh Singh (2012) "Assessment of software testing time using soft computing techniques" published in ACM SIGSOFT Software Engineering Notes, January Issue 2012. DOI: 10.1145/2088883.2088895.
[18]. Pradeep Kumar and Yogesh Singh (2013) "Comparative Analysis of Software Reliability Prediction Using Statistical and Machine Learning Techniques" published in Int. J. Intelligent Systems Technologies and Applications (IJSTA), Inder-Science Publications**,** Vol. 12, Nos. 3/4, pp: 230-253.
[19]. Singh Y, Kumar P (2010) Prediction of software reliability using feed forward neural networks. In: Proceedings of Computational Intelligence and Software Engineering (CiSE '10), Wuhan, China: 1-5. DOI: 10.1109/CISE.2010.5677251.
[20]. Kirmani, M., Wahid, A., & Saif, S. (2015). Web Engineering: An Engineering Approach for Developing Web Applications. International Journal of Software and Web Sciences, 1(12), 83-91.
[21]. Singh Y., Kaur A. and Malhotra R. (2008). Empirical validation of object-oriented metrics using discriminant analysis for object-oriented systems. Software Quality Professional, 11(1), pp: 13-24.
[22]. Singh Y., Bhatia P.K. and Sangwan O.P. (2009). ANN model for predicting software function point metric. SIGSOFT Software Engineering Notes, 34(1), pp: 1-4. DOI: 10.1145/1457516.1460352.
[23]. Singh Y., Bhatia P.K. and Sangwan O.P. (2009). Application of neural networks in software engineering: A Review. Information Systems, Technology and Management Communications in Computer and Information Science, 31, Part 6, pp: 128-137. DOI: 10.1007/978-3-642-00405-6_17.
[24]. Singh Y. (2012). Software Testing, Cambridge University Press, 2012, India.
[25]. Rauf A., Anwar S., Jaffer M.A.and Shahid A.A.(2010), Automated GUI Test Coverage Analysis Using GA, 7th International Conference on Information Technology New Generations
[26]. Rajappa V. , Biradar A.and Panda S.(2008), Effective Software Test Case Generation Using Genetic Algorithm Based Graph Theory, First International Conference on Emerging Trends in Engineering & Technology.
[27]. Ruilian zhao,shanshan lv, "Neural network based test cases generation using genetic algorithm" 13th IEEE international symposium on Pacific Rim dependable computing. IEEE, 2007.

International Journal of Computational Engineering Research (IJCER) is UGC approved Journal with Sl. No. 4627, Journal no. 47631.

Pradeep Kumar. "Soft Computing Approaches for Automated Software Testing." International Journal of Computational Engineering Research (IJCER), vol. 7, no. 10, 2017, pp. 22–34.