iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 6 **Issue:** I **Month of publication:** January 2018

DOI: http://doi.org/10.22214/ijraset.2018.1076

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor :6.887
Volume 6 Issue I, January 2018- Available at www.ijraset.com

Model for Automation of Software Testing using Genetic Algorithm to improve Software Quality

Vikash Yadav[1], Bright Keswani[2]
[1]*Research Scholar, Suresh Gyan Vihar University, Jaipur*
[2]*Professor, Dept. of Comp. Application, Suresh Gyan Vihar University, Jaipur*

***Abstract:** Test case generation is the most effort consuming part of software testing. Once the test cases were generated these were used to test the software for quality. These inputs were fed as inputs to the software to compare the observed and expected results. This comparison need to be done automatically. In this paper, a model is proposed to improve the process of software testing to improve the overall quality of software.*
***Keywords:** Software Quality, Recovery Block, Code Reuse, Software Testing*

I. INTRODUCTION

Automatic test case generation can be done by many techniques available in literature. But using these techniques and generating test cases is not enough. Comparison of results must also be done with some systematic and automatic processing. If the observed results were different than expected results than necessary action will be taken to find out the fault So automation of testing process includes two sub tasks. Firstly, generate automatic test cases based on some adequacy criteria. Secondly, Check the behaviour of program on this auto generated test case. So if test cases were chosen better, the quality will be judged better, but if test cases chosen are not enough or not of good quality then quality of program can be misjudged also.

II. RELATED WORK

Duran and Ntafos (1984) recommended a mixed final testing, starting with random testing, followed by a special value testing method (to handle exceptional cases). Ince (1986) reported that random testing is a relatively cheap method of generating initial test data. However, many errors are easy to find, but the problem is to determine whether a test run failed. Therefore, automatic output checking is essential if large numbers of tests are to be performed. They also said that partition testing is more expensive than performing an equivalent number of random tests which is more cost effective because it only requires a random number generator and a small amount of software support. The change of range for random testing has a great effect. Further they mentioned a disadvantage of random testing which is to satisfy equality values which are difficult to generate randomly.

III. PROBLEM FRAMEWORK

Every program has some particular implementation of requirement specifications. Every program has been made from some requirement specifications of that program. A program model can be viewed as a function doing mapping of some input to some output as shown in Figure 1.

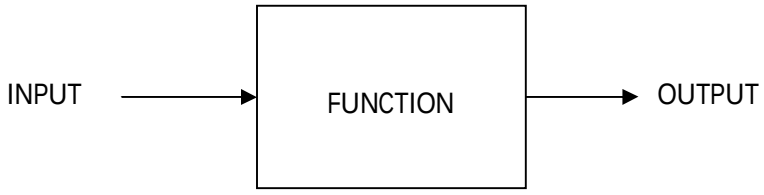


Figure 1: Program terminology

The reliability of a program depends upon the faults present in the program. The different techniques to improve it are fault avoidance, fault detection and correction & fault tolerance.

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor :6.887
Volume 6 Issue I, January 2018- Available at www.ijraset.com

The system needs to be both reliable and available. It needs to be reliable and have as few faults as possible; Fault Prevention is of paramount importance. For example, a spacecraft carrying astronauts on a round trip journey to space needs to execute flawlessly to return those astronauts safely to the ground. It also needs to be highly available so that the astronauts have access to the systems continuously.

The fundamental principle in fault-tolerance of hardware faults is the use of redundant system elements [Avizienis (1977)]. Fault-Tolerant design is a design that enables a system to continue its intended operation, when some part of the system fails, possibly at a reduced level, rather than failing completely. The use of redundant software to recover from software malfunction, however, requires special caution due to the distinctive characteristics of software. In contrast with hardware, in which physical faults predominate, software defects are time-invariant defects. Errors are produced by using the same inputs which trigger the same deficient elements of a program. Therefore, duplicate copies of a program can not remove the errors or even find the location of faults. The main cause of hardware unreliability is a random failure, which of software is its complexity. The complexity of software leads to several difficulties. These observations lead to the conclusion that if redundant software is used in an attempt to achieve software fault-tolerance, then it must not be a duplicate copy of the same software, but it must be built on same requirement specifications.

A way of designing a system that can take advantage of multiple software redundancy techniques, such as variations of Recovery Blocks and N-version programming has been described [Daniels, Kim & Vouk (1997)].

Recovery blocks were first introduced by Horning et al. [Horning et al. (1974)]. This scheme is analogous to the cold standby scheme for hardware fault tolerance. Basically, in this approach, multiple variants of software which are functionally equivalent are deployed in a time redundant fashion. An *acceptance test* is used to test the validity of the result produced by the primary version. If the result from the primary version passes the acceptance test, this result is reported and execution stops. If, on the other hand, the result from the primary version fails the acceptance test, another version from among the multiple versions is invoked and the result produced is checked by the acceptance test. The execution of the structure does not stop until the acceptance test is passed by one of the multiple versions or until all the versions have been exhausted. The significant differences in the recovery block approach from N-version programming are that only one version is executed at a time and the acceptability of results is decided by a test rather than by majority voting. The recovery block technique has been applied to real life systems and has been the basis for the distributed recovery block structure for integrating hardware and software fault tolerance and the extended distributed recovery block structure for command and control applications. Modeling and analysis of recovery blocks are described by Tomek et al. [Tomek, Muppala&Trivedi (1993)], [Tomek&Trivedi (1994)].

Recovery block is method to ensure reliability of software with alternate copies. It starts from an acceptance test, if it fails then try alternate point if again fails, then try another alternate and so on. If all the alternate tried and fails then it raises an error. It can be summarized in following block:

Ensure <acceptance test>
By
    <Primary module>
Else by
    <Alternative module>
Else by
    <Alternative module>
...
Else by
    <Alternative module>
Else error

Following figure describes the recovery block mechanism:

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor :6.887
Volume 6 Issue I, January 2018- Available at www.ijraset.com

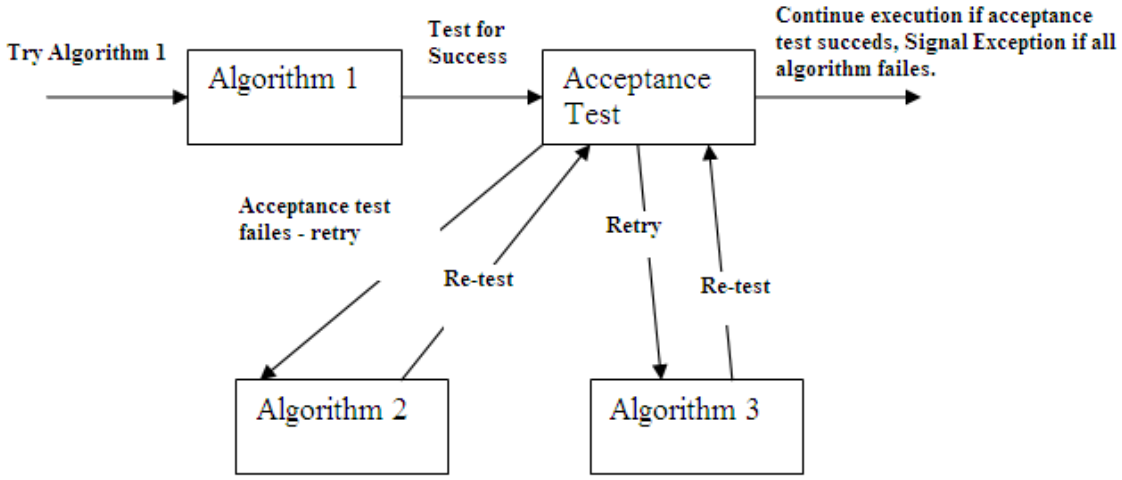


Figure 2 Recovery block

Here the alternates correspond to the variants of Program, and the acceptance test to the acceptor, with the text above being in effect an expression of the controller. On entry to a recovery block, the state of the system must be saved to permit backward error recovery, i.e., establish a checkpoint. The primary alternate is executed and then the acceptance test is evaluated to provide an acceptance on the outcome of this primary alternate. If the acceptance test is passed then the outcome is regarded as successful and the recovery block can be exited, discarding the information on the state of the system taken on entry (i.e., checkpoint). However, if the test fails or if any errors are detected by other means during the execution of the alternate, then an exception is raised and backward error recovery is invoked. This restores the state of the system to what it was on entry. After such recovery, the next alternate is executed and then the acceptance test is applied again. This sequence continues until either an acceptance test is passed or all alternates have failed the acceptance test. If all the alternates either fail the test or result in an exception (due to an internal error being detected), a failure exception will be signalled to the environment of the recovery block. Since recovery blocks can be nested, and then the raising of such an exception from an inner recovery block would invoke recovery in the enclosing block.

IV. PROPOSED MODEL FOR TESTING

The idea of recovery block mechanism is taken to propose a new model and an acceptance test is proposed as under. Every program may be viewed as a mapping of inputs to outputs. If one can build program from some requirement specifications, then one must also generate program specifications by studying the program. In other words, If P is a program which map input X to Y, then one can write a Program P' which will take Y as input and maps it to X. Now if the input of P and output of P' matches, they both will test each other by functionality. For example, if one writes a program to find the factorial (N!) of given number N, then one can also write a program to find the number N with given factorial (N!). So if one input is given to first program say 5 and 120 is the output, then this 120 can be given as an input to second program and its output can be observed, if it is 5 then both the programs do the testing of each other. If these don't match, then either of these can be wrong, so necessary actions taken. This whole process can be summarized in figure 3 as below.

This can be done in number of programs by implementing complement program of it. Two case studies are taken to study the feasibility of this model

A. Factorial of a given Number
B. Next date for a given current date

Test cases are generated using genetic algorithms and random testing. It is seen from results that genetic algorithm generate better test cases than random testing.

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor :6.887
Volume 6 Issue I, January 2018- Available at www.ijraset.com

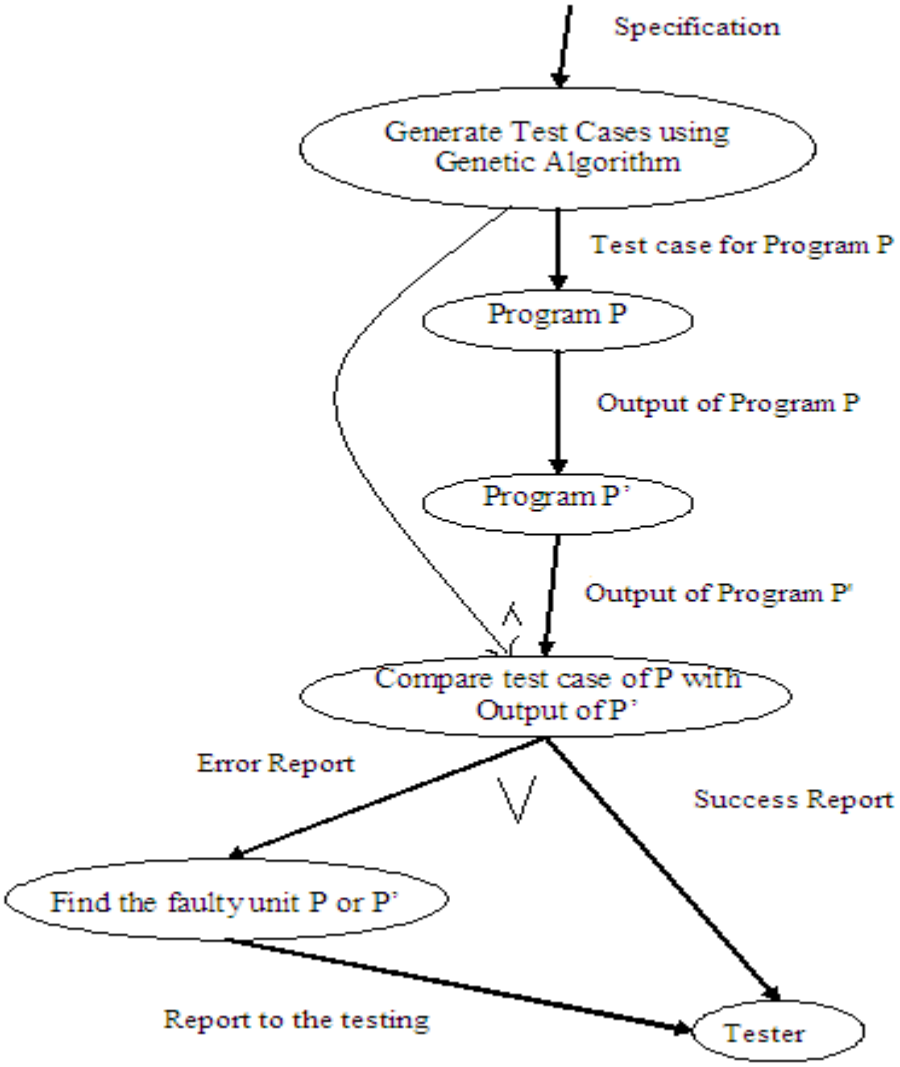


Figure 3: Proposed model based on acceptance Test

1) *Factorial of a number:* If a program can be written to find a factorial if a number Is given as an input, then one can also write a program to find the number if a factorial is given as an input to it. Coding in MATLAB is attached in Appendix-IV. So reverse program can do the job for this example:

C. Main Program To Find Factorial

1) *Input:* number whose factorial is to be calculated

2) *Output*: factorial of given number

```
Factorial()
{

        int i=0, fact=0;
        for i = 1 to num
                fact = fact * i;
        end

}
```

D. Second Program To Find Number Whose Factorial Is Given

1) Input : factorial of given number

2) Output : number whose factorial is to be calculated

```
NumofFactorial()
{

        intnum = 1, tfact =1;
        for c = 1 to fact
                tfact = tfact * c ;
                if (tfact == fact)
                        break;

                end
        end
```

So a number can be passed to Factorial function to find out the factorial of it, and then whatever is the output of factorial function is can be again passed as input to Num of Factorial function. It will generate some output, and then this output will be checked with original input, if they are same, it means that both the programs written are correct. As they do the testing of each other.

3) Next date function: This is the function which returns next date of the entered date. This can also be used in Recovery block, as if one can write a program to find the next date of it, then a program for finding the previous date can also be written. Coding in MATLAB is attached in Appendix-III.

E. Main Program To Find Next Date

1) Input : a date

2) Output : next date of input date

```
Next Date()
{
        Read current date;
        check if month is in range of 1-12;
        check if date is within range 1-31;
        check if year is within range 1951-2050;
        if any check fail then return to the input of current date with error message. else
                Calculate nextMonth;
                Calculate nextDay;
                Calculate nextYear;
        Return NextDate;
        date with error message. else
                Calculate prev Month;
                Calculate prev Day;
                Calculate prev Year;
```

So a date can be passed to Next Date function to find out the next date of it, and then whatever is the output of Next Date function is can be again passed as input to Previous Date function. It will generate some output, and then this output will be checked with original input, if both dates are same, it means that both the programs written are correct. As they do the testing of each other.

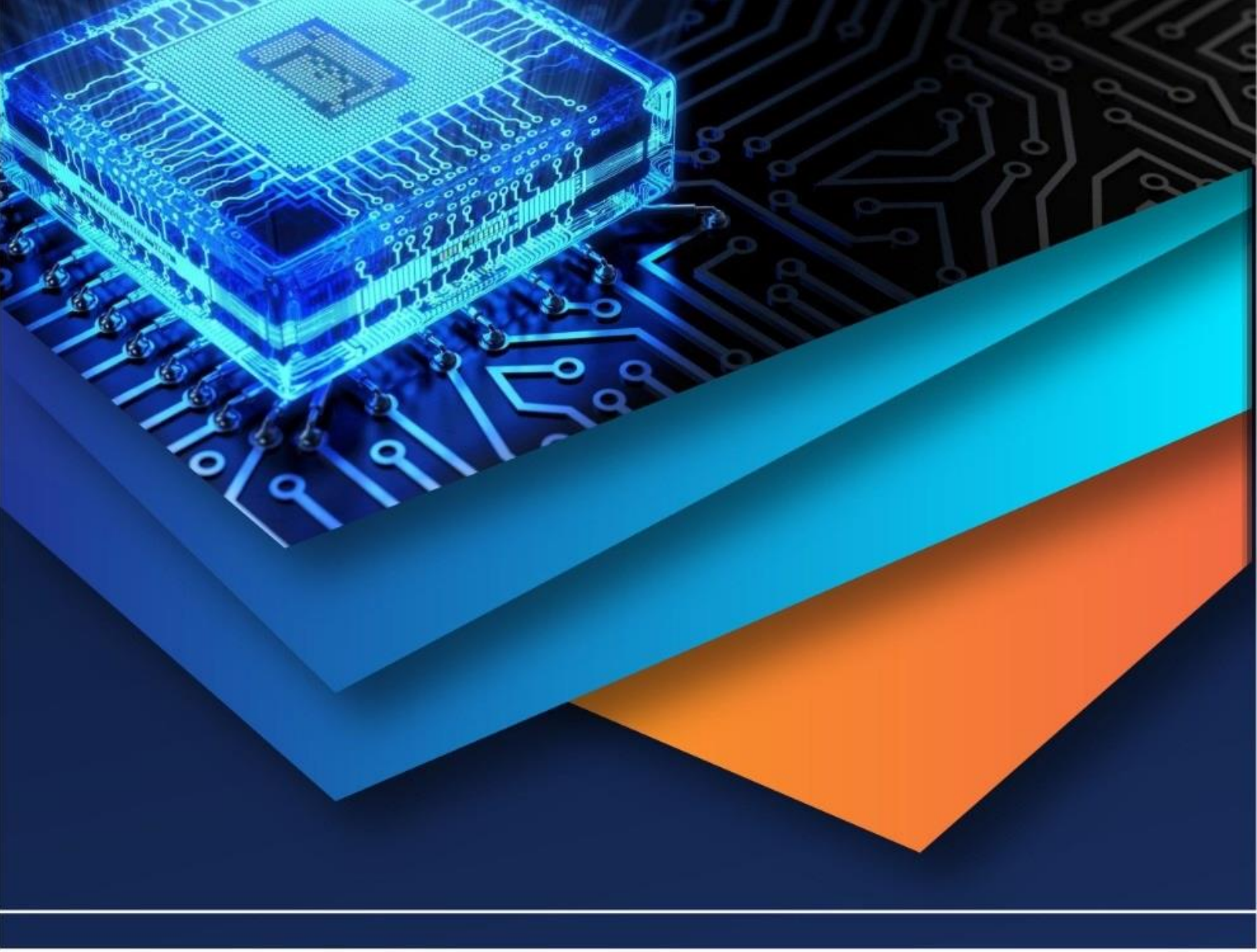
V. CONCLUSION

The idea of proposed model was taken from recovery block technique. Recovery blocks were first introduced by Horning and his team mates. This scheme is analogous to the cold standby scheme for hardware fault tolerance. Basically, in this approach, multiple variants of software which are functionally equivalent are deployed in a time redundant fashion. An *acceptance test* is used to test the validity of the result produced by the primary version. If the result from the primary version passes the acceptance test, this result is reported and execution stops. If, on the other hand, the result from the primary version fails the acceptance test, another version from among the multiple versions is invoked and the result produced is checked by the acceptance test. The complementary

version of program can do the task. Two programs were taken for experiments in this chapter. Test cases were generated using Genetic Algorithm and random testing for them and the acceptance test are defined for them.

REFERENCES

[1] Avizienis A., "Fault-Tolerance and Fault- Intolerance: Complementary Approaches to Reliable Computing," Proc. 1975 Int. Conf. Reliable Software, pp. 458-454.
[2] Daniels, F., K. Kim and M. Vouk. "The Reliable Hybrid Pattern: A Generalized Software Fault Tolerant Design Pattern." Presented at PLoP 1997, Monticello, IL. September 1997. [http://hillside.net/plop/plop97/Workshops.html]
[3] Duran J. W. and Ntafos S. C.: 'An Evaluation of Random Testing', IEEE Transactions on Software Engineering, Vol. SE-10, No. 4, pp. 438-444, July 1984
[4] Duran, J. W. and Ntafos S., 'A report on random testing', Proceedings 5th Int. Conf. on Software Engineering held in San Diego C.A., pp. 179-83, March 1981.
[5] Goldberg, D.E., (1994) Genetic and Evolutionary Algorithms Come of Age, Communications of the ACM, Vol.37, No. 3, pp 113.
[6] Horning J. J., Lauer H. C., Melliar-Smith P. M. and Randell B., "A Program Structure for Error Detection and Recovery." Lecture Notes in Computer Science, 16:177-193, 1974.
[7] Ince, D. C.: "The automatic generation of test data", The Computer Journal, Vol. 30, No. 1, pp. 63-69, 1987.
[8] Knight, J. C. and N. G. Leveson, "A reply to the criticisms of the Knight &Leveson experiment," SIGSOFT Softw. Eng. Notes 15, 1 (Jan. 1990), 24-35.
[9] Offutt J. and Hayes J., "A semantic model of program faults". In International Symposium on Software Testing and Analysis (ISSTA 96), pages 195{200. ACM Press, 1996.
[10] Saridakis, T. "A System of Patterns for Fault Tolerance", Proceedings of EuroPLoP 2002, KlosterIrsee, Germany, July 2002, pp 535-582.
[11] Staknis, M. E.: 'Software quality assurance through prototyping and automated testing', Inf. Software Technol., Vol. 32, pp. 26-33, 1990
[12] Tomek L., Muppala J. and Trivedi K. S.,"Modeling Correlation in Software Recovery Blocks". In IEEE Transactions on Software Engineering (special issue on Software Reliability), Vol. 19, No.11, November 1993, pp. 1071-1086.
[13] Sonia Bhargava, Bright Keswani, "Generic ways to improve SQA by meta-methodology for developing software projects", International Journal of Engineering Research and Applications, Vol. 3, Issue 4, pp 927-932, July 2013.
[14] Tomek L. and Trivedi K. S., "Analyses Using Stochastic Reward Nets", In Software Fault Tolerance, ed. M. Lyu, John Wiley & Sons, 1994.
[15] Torres-Pomales, W., "Software Fault Tolerance: A Tutorial, Technical Report," Report No. NASA-2000-tm210616, 2000.



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY