Available Online at www.ijcsmc.com

International Journal of Computer Science and Mobile Computing

A Monthly Journal of Computer Science and Information Technology

ISSN 2320–088X

IJCSMC, Vol. 4, Issue. 7, July 2015, pg.167 – 173

RESEARCH ARTICLE

Enhancement in Debugging Technique to Find Bugs in System Software

Maneela Sharma[1], Ms. Harsimranjeet Kaur[2]

M.Tech Student[1], CEC Landran, PTU
m.sharma8790@gmail.com[1]

Assistant Professor[2]
CEC LANDRAN, Punjab Technical University
cecm.cse.sim@gmail.com[2]

Abstract: Debugging is the process of finding and fixing the bugs or faults. In the debugging process we find out the bugs or errors in the computer program. Many bugs are discovered through software testing but software testing rarely eliminates every bug in the program. After debugging process when the faults still present in the program that is called imperfect debugging. In this paper we propose a mutation based genetic algorithm (GA) for the detection of imperfect debugging and next we will show the enhancement of genetic algorithm (GA) to detect the imperfect debugging in the software.

Keywords: Debugging, imperfect debugging, genetic algorithm, mutation.

1. Introduction

Software engineering is all about sequence of steps to produce the software, from its initial stage to its final stage. A software engineering is related to all the aspects that are used in the software production or create the software. Software is a generic term that is used for organizing the data and instructions that are collected to develop it. The software is divided into the two categories: System Software and the Application Software. The system software is used to manage the hardware components, so that other software or user sees it as a functional unit. The software contains the operating system and some more utilities like disk formatting, file managers, display managers, etc. The application software used for accomplished the specific tasks. Application software may or may not contain the single program. Software is the program or set of programs. As in

software many things are includes: as it consists of the programs, the complete documentation of that program, the procedure that is use to set up the software and the various operation of the software system. Software Engineering is a profession to provide high quality software products to its customers. It is an application of systematic, disciplined approach to development, operation, maintenance of software. Software consists of seven phases and these phases are called Software Development Life Cycle.

1.1 Debugging: A software development process is use to translate the software product, in which the customer translate all the needs to the developers that what kind of changes a customer required. In computers, debugging is locating and fixing bugs (errors) in compute program code. Debugging is the methodical process of finding and reducing the number of bugs or defects in the computer program. [3].

1.2 Imperfect Debugging: In order to ensure the high reliability, it is essential for software to undergo a testing phase, during which faults can be detected and corrected by the debuggers. The testing resource allocation during this phase, which is usually depicted by the testing efforts function, influences not only the fault detection rate but also the time to correct a detected fault. In addition testing can be imperfect due to complexity of software or incomplete understating of the software. The testing team may not be able to remove/correct faults or defects perfectly and that faults may remain in the software resulting in the phenomenon of imperfect debugging [4].

***1.3 Imperfect Debugging Effects*:** It is always recognized that debugging processes usually imperfect. If debugging of software is imperfect then, more faults/defects may be introduced and detected into the software under test. Software faults are not completely removed because of the difficulty in locating them or because new faults might be introduced [5].

2. Review of Literature

In this paper [3] new model has been developed to catch the effect of faults generated by multi release in the software. This model takes into account the remaining faults of previous release and faults of current release. Many imperfect debugging the process are not successful. The complete understanding of tester team about the software the tester team may not able to remove the faults perfectly and original faults remain in the software product that is imperfect debugging. In this model the effect of both type of imperfect debugging during the testing phase are incorporated in the developed multi release model.

In this paper [4] proposed the mathematical or optimization techniques for requirement selection. The main purpose of next release problem is to select the requirement from all subsets of requirements. The Pareto tool and other mathematical tools used for the solution of next release problem. Simulated Annealing techniques such as fitness function, cooling schedule, initial temperature and parameters applied for solution of next release problem. The genetic algorithm is also a research tool. That is also used in next release problem. In this paper the author also describes about ants colony optimization algorithm and practical application. This application tells how practical effort helps the developers for requirement selection for next release problem. So this paper proposed the mathematical techniques for requirements selection means next release problem.

In this paper [5] conducted study on software optimal release policy and reliability growth modelling. These software reliability growth models are less helpful for their developers. Software reliability growth models needs to put more testing efforts for make the models more reliable. Therefore in this paper for constructing software reliability growth model based on non homogeneous Poisson process a scheme proposed. Reliability growth model are based on non homogeneous Poisson process and various functions that are performed by reliability growth model are based on non homogeneous Poisson process. Software reliability models based on NHPP which incorporates generalized exponential distribution during the testing phase. In this paper parameters of model are estimated by Least Square and the maximum likelihood methods. In this model many software systems has been used and optimal release policy also described.

In this paper [6] introduced about allusion to the flaws in software cost model and optimal release policy, inadequate consideration for real debugging, a cost model and optimal release policy for SRGM (Software Reliability Growth Model) incorporating imperfect debugging is proposed. A SRGM is presented, based on incomplete debugging, introduction of new faults and Testing Effort. It is verified to describe real testing process well by actual failure data set and has better performance as compared to other models. Based on the proposed SRGM, a formulation of cost function is also established especially considering the impact of imperfect debugging on cost. Furthermore, the optimal release policies given limited reliability objective and the uncertainty in actual total cost exceeding expected one are developed and elaborated. Finally, a numerical example and related data analyses are illustrated and parameter sensitivity analysis is performed.

In this paper [7] recognized that the debugging processes are usually imperfect. Software faults are not completely removed because of the difficulty in locating them or because new faults might be introduced. Hence, it is of great importance to investigate the effect of the imperfect debugging on software development cost, which, in turn, might affect the optimal software release time or operational budget. In this paper, a commonly used cost model is extended to the case of imperfect debugging. Based on this, the effect of imperfect debugging is studied. As the probability of perfect debugging, termed testing level here, is expensive to be increased, but manageable to a certain extent with additional resources, a model incorporating this situation is presented. Moreover, the problem of determining the optimal testing level is considered. This is useful when the decisions regarding the test team composition, testing strategy, etc., are to be made for more effective testing.

3. Reliability Models

The reliability improvement process is called reliability growth. Software reliability models are playing very important role to judge the reliability of the product or to judge the later faults or imperfects in the software product or thing when it is deliver to the customers. Reliability models are used to judge the reliability of software product that the software product is released in the market how much reliable for the customer [6]. Reliability means consistency, stability, trust ability. Reliability models can be classified into two categories:

(1) Static models
(2) Dynamic models

Static models use other situations or events of the product or program parts to estimate the numbers of faults in the software [1]. Dynamic models usually based on statistical distributions. These models use the steady development faults pattern to estimate the end product reliability. Dynamic software reliability models can be further classified into two categories [3]:

(a) The models that are entire development process.
(b) The models that are back-end testing phase.

3.1 Jelinski Moranda Model: It is time between failures model which comes under the categories earliest models. It assumes fix time is negligible and the fix time should be perfect for each failure. So the software product's failure rate improves by the same amount at each fix.

3.2 Little wood models: It is also time between failure models. It is same as Jelinski Moranda model but it assumes that different faults have different sizes, thereby contributing unequally to failures. The introduction of fault size provides more realistic to model assumption. It developed some another models like little wood non homogeneous Poisson process [9]. The Jelinski Moranda model assumes perfect debugging. But this is not always possible because in the process of fixing a fault, new faults may be injected. Therefore Goel Okumoto proposed an imperfect debugging model to overcome the limitation of the assumption.

Some models are also come under the class of faults count models like as Goel Okumoto Non homogeneous Poisson process model, Musa-Okumoto Logarithm [9].

3.3 Poisson Execution Time Model, Delayed S and Inflection S Models: Non homogeneous Poisson process model is related with modelling the number

of failures in given testing period. Musa-Okumoto Logarithmic Poisson Execution time model is similar to NHPP model. In model also the number of failure are observed in a certain period of time interval [11].

3.4 Goal-Okumoto non homogeneous Poisson process model: Goel and Okumoto propose that the cumulative number of failure observed at time t, N (t), can be modelled as a non homogeneous Poisson process as a Poisson process with a time- dependent failure rate

3.5 Musa- Okumoto Logarithmic Poisson Execution Time Model: It is similar to non-homogeneous Poisson process model. It is also assumes to be non homogeneous Poisson process. It efforts to find the later faults those have a smaller effects on the software reliability than earlier ones.

***3.6 The Delayed S and infection S Models*:** In 1983 Yamada et al. proposed a new model known as The Delayed S and Infection S Models in which the observed growth curve of the cumulative number of detected defects is S- Shaped. The model is based on the non homogeneous Poisson process, but with a different mean value function to reflect the delay in failure reporting. In 1984, new S- shaped reliability growth model known as infection S model that was proposed by Ohba. This model describes a software failure detection phenomenon with a mutual dependency of defected defects [9].

4. Proposed Methodology

Debugging is the methodical process of finding and reducing the number of bugs or defects in the computer program. There are 15 factors in imperfect debugging which are responsible for faults and warning. In this work, we have to decrease the effect of the imperfect debugging. So overcome this effect enhanced in genetic algorithm has been done. The enhancement will be based on function importance and number of functions associated. The crossover value will be taken on function importance.

Algorithm

1. Generate initial population of chromosomes
   Let it be N={c1,c2…..cn)
2. Assign Chromosomes to test suites
   Set Test case=No of chromosomes(N)
3. Apply fitness function
   Apply fitness function= total number of functions associated with changes made in function
4. Select best chromosomes according to based fitness function
5. Genetic Operators Applied
   Do for selected Chromosome(s)
6. while (iter < MAXITER) f
7. select (x1; x2)
8. y<- Crossover(x1; x2)
9. y<- Mutate(y)
10. y<- GreedyRepair(y)
11. if (y= x for any x = P) {
12. goto 5
13. }
14. find worst chromosome xmin = P
15. replace xmin with y
16. evaluate y
17. if (y better than x){
18. x<- y }
19. iter<- iter + 1
20. }
21. Optimization of solution checked.
    If (solution!= feasible)
    Goto STEP 5
    Else END.

In this work, enhancement will be proposed in the tradition genetic algorithm for imperfect debugging. In the previous algorithm the chromosome value are given to the algorithm and on the basis of chromosome values base fit value will be calculated check imperfect debugging effects. In the enhanced algorithm, chromosome values are given as the input to the algorithm and to select the best fit value iterative technique has been followed in which the error is calculated every time . The best fit value will be changed from the chromosome value until the error will be minimized.

4.1 Parameters to check effect of imperfect debugging and testing

1. Complexity

2. Lack User Support

3. More Work Pressure

4. People Do Not Get Work According to Expertise

5. Planning to Catch Up Later

6. Omitting unnecessary Tasks

7. Lack Efficient Management

8. Weak Efforts for testing

9. Imperfect Based Development

10. Externally Supplied Components Influence

11. Uncontrolled Employee Problems

12. Short changed Quality Assurance

13. Undermined Motivation

14. Adding People Late to Project

15. Coding Like Hell Programming

5. Experimental Results

The proposed technique is implemented on MATLAB.

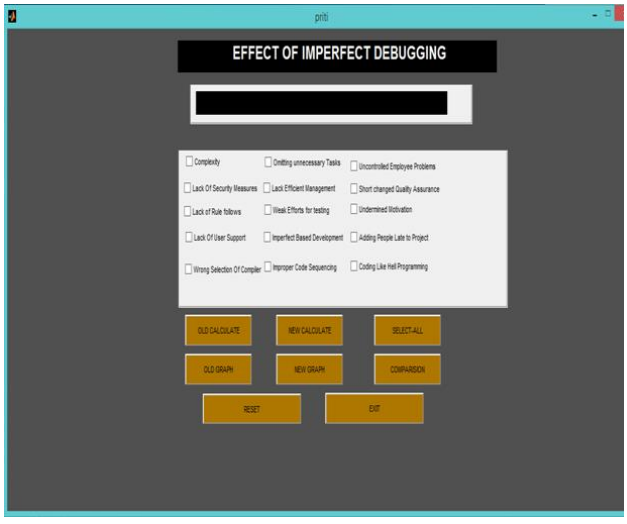

Fig 1.1: Default tool

As illustrated in figure 1.1, the tool is developed to test the imperfect debugging of the software. In the tool various factors are considered which will effects normal behaviour of application.

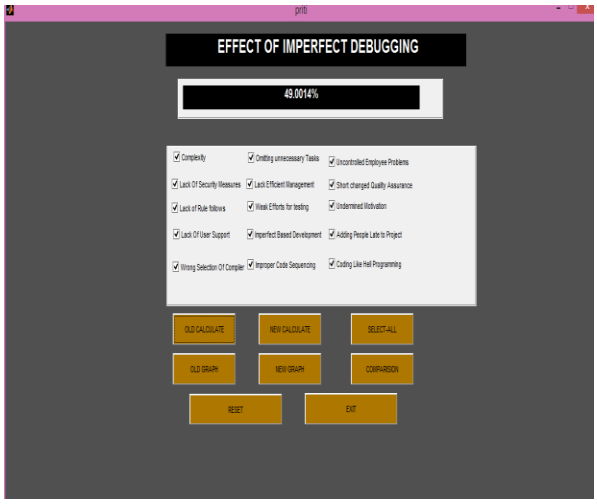

Fig 1.2 Effect of factors (with old algorithm)

As illustrated in figure 1.2, the tool is developed to test the imperfect debugging of the software. In the tool various factors are considered which will effects normal behaviour of application. The fifteen factors are selected which gave output with the old algorithm as 49.0014 %

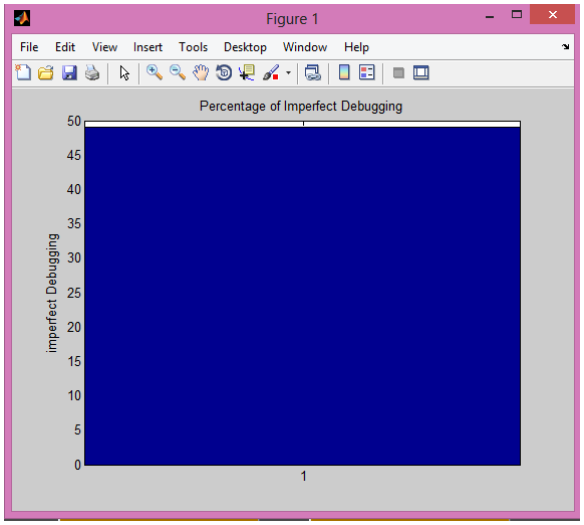

Fig 1.3: Bar Graph

As illustrated in figure 1.3, the tool is developed to test the imperfect debugging of the software. In the tool various factors are considered which will effects normal behaviour of application. The fifteen factors are selected which gave output with the old algorithm as 49.0014 % .

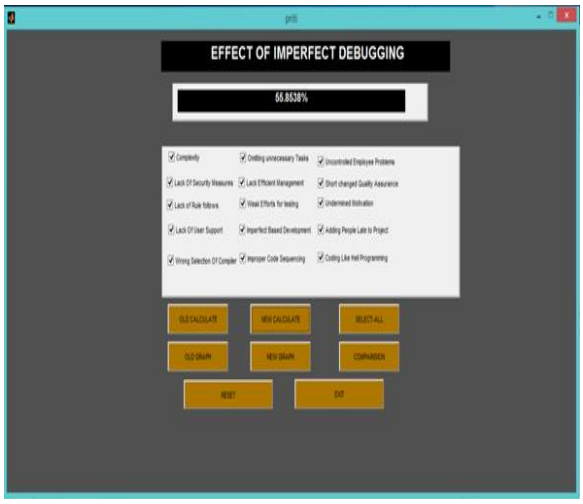

Fig 1.4: Effect of factors (with enhanced algorithm)

As illustrated in figure 1.4, the tool is developed to test the imperfect debugging of the software. In the tool various factors are considered which will effects normal behaviour of application. The fifteen factors are selected which gave output with the enhanced algorithm as 55.8538 %

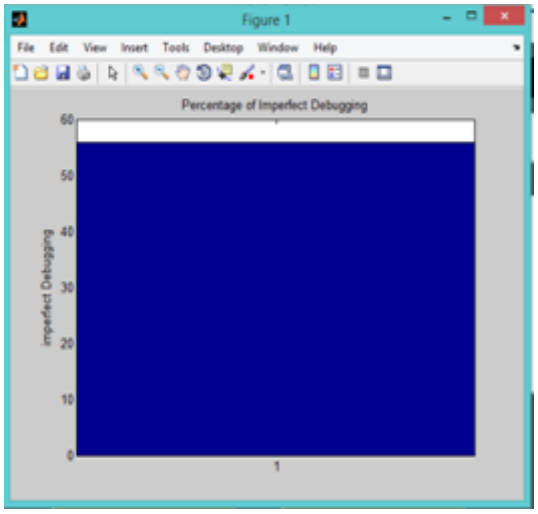

Fig 1.5: Bar Graph

As illustrated in figure 1.5, the tool is developed to test the imperfect debugging of the software. In the tool various factors are considered which will effects normal behaviour of application. The fifteen factors are selected which gave output with the enhanced algorithm as 55.8538 % .

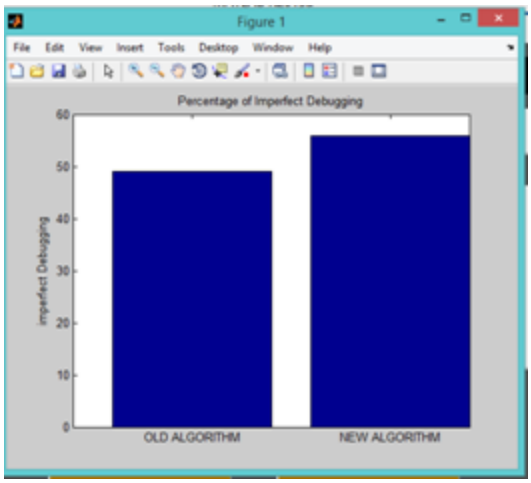

Fig 1.6 Comparison graph enhanced and existing technique

As illustrated in figure 1.6, the comparison is made between existing and enhanced technique and bar graphs are used for comparison. From the above figure it proves that enhanced technique is more efficient than existing technique.

6. Conclusion

During testing the testers do not know that they have covered how much part of the testing phase. Throughout the testing coverage testers come to know about how much part they have covered of testing phase. This study will be beneficial in the software reliability growth models field. By this study we will incorporate the effect of imperfect debugging in a multi release. The second issue test factors that are related to multi release concept, we will try to bring minor improvements in it. In this paper, genetic algorithm has been used to find out errors in the design of the software.

Reference

[1] Kapur P.K., Pham H., Fellow, Aggarwal Anu G., and Kaur Gurjeet, "Two Dimensional Multi-Release Software Reliability Modelling and Optimal Release Planning", IEEE Transactions on Reliability Vol.61,No 3 September 2012.

[2] Aggarwal Anu. G, Kapur P. K. and Garmabaki A. S. ", Imperfect Debugging Software Reliability Growth Model for Multi Release," proceeding of the

5th National conference: INDIA Com-2012, Computing for National Development March 10-11.2011.

[3] Singh Jagvinder, Singh Ompal, Aggarwal Deepti, Adarsh. Anand and Singh Inderpal," A flexible reliability growth model for various releases of software under The influence of testing resources imperfect debugging," (copyright © 2011NLSS, Vol. 2)2, July 2012, pp 23-35.

[4] Sagrado Jose del., Isabal, Aguila M. del., Orellana Francisco j. and Tunez S.," Requirement selection: Knowledge based optimization techniques for solving the next release problem," Dpt. Language and computation university of Almeria, spain

[5] Quadri S. M. k., Ahmad N., Peer M. A.," Software optimal release policy And reliability growth modeling," Proceeding of the 2nd National conference ; INDIA Com-2008, computing for nation development, Feb 08-09, 2008.

[6] Ce ZHANG et.al, "A Study of Optimal Release Policy for SRGM with Imperfect Debugging", Journal of Engineering Science and Technology Review 6 (3) 111-118,2013

[7] Min Xie; Bo Yang, "A study of the effect of imperfect debugging on software development cost," Software Engineering, IEEE Transactions on , vol.29, no.5, pp.471,473, May 2003 doi: 10.1109/TSE.2003.1199075

[8] Hoang Pham, "An Imperfect-debugging Fault-detection Dependent-parameter Software", International Journal of Automation and Computing, 04(4), October 2007, 325-328

[9] Dr. Satya Prasad Ravi N.Supriya and G.Krishna Mohan, "SPC for Software Reliability: Imperfect Software Debugging Model", IJCSI International Journal of Computer Science Issues, Vol. 8, Issue 3, No. 2, May 2011 ISSN (Online): 1694-0814

[10] K. Tokuno and S Yamada, "An imperfect debugging model with two types of hazard rates for software reliability measurement and assessment", Elsevier Mathematical and Computer Modelling Volume 31, Issues 10–12, May–June 2000, Pages 343–352

[11] Hyunsook Do, Siavash Mirarab, "The effect of time constrainton test case prioritization" IEEE TRANSACTION ON SOFTWARE ENGINEERING ,VOL.36 , ,IEEE, 2010

[12] Gray Y.,Chnel, Jamie Rogers, "Arranging software test case through an optimization method", IEEE Chung Yuan Christian University, Industrial & Systems. Engineering, Chung Li, Taiwan University of Texas – Arlington, Industrial & Mfg. Systems Engineering, Arlington, TX – USA. 2010

[13] Shifa-e-Zehra Haidry and Tim Miller, "Using Dependency Structures for Prioritisation of Functional Test Suites"IEEE, 2012

