ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)
Vol 5, Issue 3, March 2018

Software Reliability Growth Model using Testing coverage, Function point and Test point analysis

[1] Amol K. Kadam, [2] Dr. Shashank D. Joshi,[3] Sachin B. Wakurdekar
[1][2][3] Bharati Vidyapeeth Deemed to be university College of engineering, Pune

Abstract – **Testing is an important activity to ensure software quality but long time testing may not insure bug free software and high reliability. Optimum amount of code also need to be covered to make sure that software is of good quality and high reliability. In these proposed Software Reliability Growth Model analyze all codes files of the project. In this model every code file of the project is analyze and provide the suggestion to the user for improving performance of the system. Also this model calculate the cost of the project that cannot be calculate at existing software reliability growth model. This model focused on testing time, testing coverage, functional point analysis and test point analysis to increases the reliability of software, calculate software cost and optimize the software maintenance cost.**

Keywords: Testing coverage, Functional point analysis, Test point analysis, SRGM, COCOMO.

INTRODUCTION

Testing is a crucial activity to make sure code quality. Huge organizations will have many development groups with their product being take a look at by full test groups. Take a look at team managers should be able to properly set up their schedules associated resources and estimates for the needed take a look at execution effort will be an extra criterion for take a look at choice, since effort could be restrictive in follow. An honest take a look at execution effort estimation approach will profit each tester managers and code comes. There's estimation model associated an expertise primarily based approach for take a look at execution effort. The probability of failure-less operation in a specified environment in a specific period of time under specific conditions is called as Software Reliability. Software Reliability Growth models (SRGM) is developed for the estimate software reliability measures such as number of remaining faults, software failure rate and software reliability. Software testing can be defined as a process to detect faults in the entire developed computer software which falls in the category of Software development life cycle (SDLC) phases. Software testing helps us to detect the probable faults and errors in the developed software. Testing of the software for longer time does not ensure bug free software with higher reliability. Optimum amount of code also needs to be covered to make sure that the software is of good quality. It is hard to remove the entire faults in the software due to its complex nature. This is also termed as imperfect debugging. Error generation is defined as phenomena in which remaining faults in the software leads to further generation of faults. Test estimation consists of the estimation of effort and value for a selected level of testing, exploitation numerous ways, tools, and techniques. The wrong estimation of testing effort typically ends up in associate inadequate quantity of testing, which, in turn, will result in failures of software package systems once they're deployed in organizations. Estimation is that the most crucial activity in software package testing, associated an ineluctable one, however it's typically performed hurriedly, with those liable for it simply hoping for the simplest. Testing is directed toward inputs and program parts wherever errors are a lot of possible. The main target of testing is on finding defects, and defects typically often found abundant quicker by totally different testing attributes. It's vital to balance the relationships between effort, schedule and quality. It's wide accepted that merely estimating one in every of these aspects while not considering the others can lead to phantasmagorical estimations. Classical estimation models are established supported linear or non-linear multivariate analysis, that incorporate mounted input factors and stuck outputs.

OBJECTIVE OF THE RESEARCH WORK:

- To improve performance & reduce maintenance cost.
- Estimate Software Cost.
- Check the efficiency of development activities
- Quality and Testability of the test object
- Interdisciplinary Research Project
- Industrial Consultancy
- Academic Research Activities

ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)
Vol 5, Issue 2, March 2018

In this research work we have developed Software Reliability Growth Model that contain Testing time and testing coverage, Function Point Analysis and Test Point Analysis. It was an attempt to overcome difficulties associated with lines of code as a measure of software size, and to assist in developing a mechanism to predict effort associated with software development.

METHODOLOGY:

Module I: Testing time and Testing coverage, Function Point Analysis
Enhanced Non Homogenous Poisson process (EHPP):
Testing Time: Either the CPU time or calendar time is referred as Testing time.
Testing Coverage: The prediction of the software reliability is ensured through testing coverage.

A software developer make use of Testing Coverage to evaluate the quality of the tested software and also helps to determine the additional effort required for improving the reliability of the software.
Threshold value: We have to set threshold values in testing coverage and function point analysis. This threshold values we have to set from the reference of reputed journal papers and industry experts in various companies.
The threshold value is interpreted based on previous projects experience and historical information. While considering the threshold value, benchmarks designed by industries also taken into grant. From the team experience and various processes involved the threshold is monitored and updated.
Basic Testing Coverage Measures:
1. Statement Coverage: Number of lines processed in the program. If number of the lines are exceeded more than threshold range then giving advice to user.
2. Path Coverage: It indicates number of viable paths that exist in the code and also find the inheritance tree.
3. Decision / Condition Coverage: It tells whether Boolean expressions tested in control structures are true or false. If Boolean expressions are more than threshold range then giving to advice
4. Procedure Coverage: It gives number of procedures determined by the testing Software reliability growth models (SRGM). In that also giving advice to user when no of the procedures and functions are more than threshold value.
So We have to find out reliability of the software and also giving advice for increases reliability of the software.

Function Point Analysis:
FPA is the method of calculating the size of the software by using the complexity of software functionalities using the function points. Then function point is used for the estimating the effort to develop it.
Calculate Size of Project: In this phase it calculates the lines of code, blank spaces as well as comments in the project. If the number of lines per class is greater than threshold value then it advice for splitting of the class.
Calculate Number of Object in Class: Total number of object in class are identified For the project given to system then after clicking on calculate number of object of class it calculates number of attributes of the class. if the total number of attributes in class are more than threshold range then advice for splitting the class.
Calculate Number of Methods: In this module it calculates how many methods in each class of the project. If method values don't match between threshold range 3-7 then provide suggestion for splitting the methods.

PROPOSED MODEL:

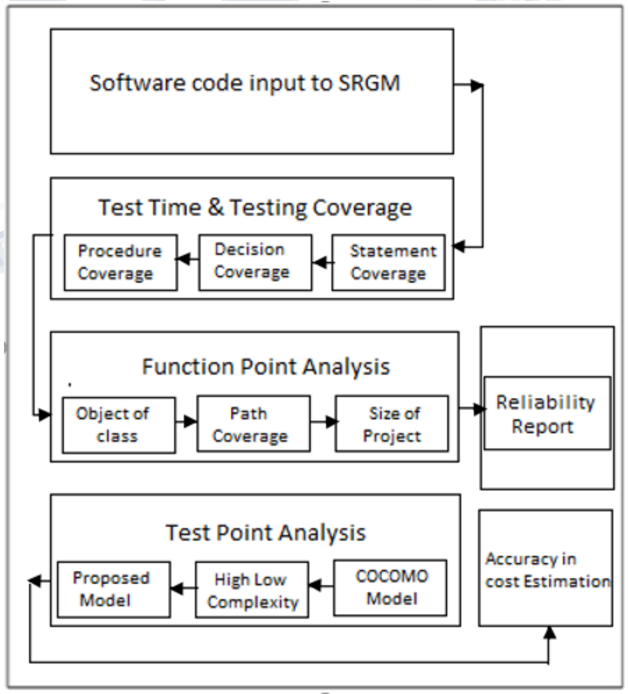


Fig: Architecture of Proposed SRGM

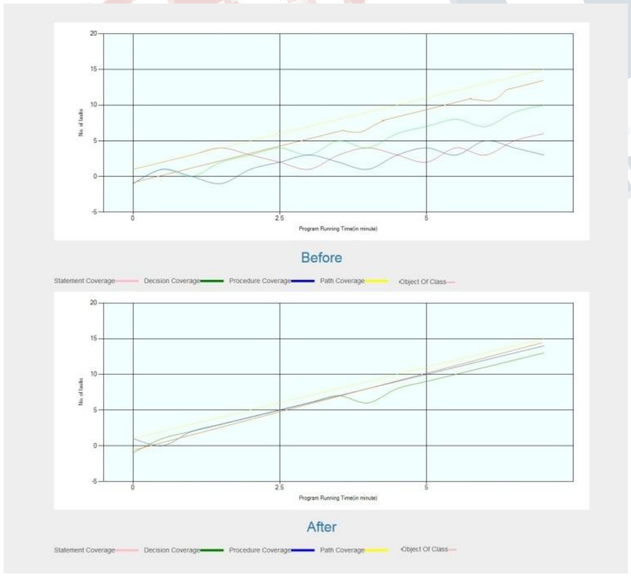
Level 1-Input to SRGM: All files from project code are been read line by line in temporary data structure for future processing.

ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)
Vol 5, Issue 2, March 2018

Level 2-Test Time &Testing Coverage: Firstly statement coverage is been done which analysis no of lines of code, blank lines. Secondly loop and control structure analysis is been done in decision coverage finding numerical value of metrics. Threshold range is been initialized for all classes. Finally in procedure coverage weighted methods are been found with function description.
Level 3- Function Point Analysis: This level in depth analysis of code is been done finding software complexity with functionality analysis. Complete effort are been computed in function point analysis. Number of objects that are been initialized in complete code. This values are been compared with threshold for number advisable objects . Additionally attributes in all class are been computed. Finally methods functions written in single class are been found, this values are also computed against threshold set. Finally if any threshold are been violated then suggestion are been reported.
Level 4-Test Point Analysis: Mainly we focused on accuracy of estimating the cost of the software. In that we provide the complexity of that software to Basic COCOMO Model. Complexity is mainly depends upon five parameters like Methods, Boolean Expressions, Objects, Line of code, Procedure coverage.

RESULTS & DISCUSSION:



Graph 1: Before and After using Threshold values (Difference in complexity)

In above graphs 1 before using threshold values the zigzag lines are more so we analyze the complexity of that software is high and after using threshold values and modified the project according to our suggestions find out the zigzag lines are less compare to before using threshold values, So we declare that when developer develop the software using our threshold range then definitely reliability of that software is increases.

Module II: TEST POINT ANALYSIS

Mainly we focused on accuracy of estimating the cost of the software. In that we provide the complexity of that software to Basic COCOMO Model. Complexity is mainly depends upon five parameters like Methods, Decision coverage, Objects, Line of code, Procedure coverage

Proposed SRGM for estimating cost: E=ai (KLoC) (bi)* Complexity
where E is the effort applied in person-months,
KLoC is the estimated number of thousands of delivered lines of code for the project,
ai, bi, ci di are Constants.

Software Project	a_i	b_i	c_i	d_i
Organic	**2.4**	**1.05**	**2.5**	**0.38**
Semi-Detached	**3.0**	**1.12**	**2.5**	**0.35**
Embedded	**3.6**	**1.20**	**2.5**	**0.32**

Table I: Constant Values from COCOMO

Complexity : Low:0.75, High: 1.25.
We have to analyse more than 30 project and decide constant values for high and low complexity. That values are mainly depends upon KLoC of that project shown in following table

KLoC	Parameters	Range	
		Low	**High**
	Methods	**0-50**	**<50**
	Decision	**0-30**	**<30**

ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)
Vol 5, Issue 2, March 2018

2-50	Coverage		
	Objects	0-45	<45
	Line of code	0-5000	<5000
	Procedure Coverage	0-20	<20
51-300	Methods	51-120	<120
	Decision Coverage	31-75	<75
	Objects	46-100	<100
	Line of code	5000-51000	<51000
	Procedure Coverage	21-50	<50
Above 300	Methods	121-180	<180
	Decision Coverage	36-100	<100
	Objects	101-150	<150
	Line of code	3 lacks	<3 lacks
	Procedure Coverage	51-100	<100

TableII: Set values for Complexity

Accuracy in Cost:
Basic COCOMO do not find the actual complexity, it's find out the efforts using KLoC so it don't know how much complexity in that project/software but our proposed model analyze complexity and provide that complexity to Basic cocomo then definitely we have to achieve accuracy in efforts, development time, staff size and productivity.
Following calculations shows the comparison between Basic cocomo and our proposed model:
Low Complexity:
Basic COCOMO Model: E=a_i (KLoC) $^{(bi)}$

KLOC= 3227/1000

= 3.327

Efforts: E[i]= a[i]* (KLoC) $^{(bi)}$

= 2.4* 3.327 $^{(1.05)}$

= 08.21 Man-Month

Development: D[i]= c[i]* E[i] $^{(di)}$

= 2.5*08.21$^{.38}$

= 5.56 Months

Productivity: P[i]=KLoC/E[i]

=3.327/08.21

= 0.392 Per month

Proposed Model: E=a_i (KLoC) $^{(bi)}$*Complexity

KLOC= 3327/1000

= 3.327

Efforts: E[i]= a[i]* (KLoC) $^{(bi)}$ *Complexity

= 2.4* 3.327 $^{(1.05)}$ * 0.75

= 06.15 Man-Month

Development: D[i]= c[i]* E[i] $^{(di)}$

= 2.5*06.15$^{.38}$

= 04.98 Months

Productivity: P[i]=(KLoC/E[i])

=(3.327/06.15)

= 0.529 Per month

<u>High Complexity:</u>

- **Basic COCOMO Model: E=a_i (KLoC) $^{(bi)}$**

KLOC= 7749/1000

= 7.749

Efforts: E[i]= a[i]* (KLoC) $^{(bi)}$

= 2.4* 7.749 $^{(1.05)}$

= 20.62 Man-Month

Development: D[i]= c[i]* E[i] $^{(di)}$

= 2.5*20.62$^{0.38}$

= 7.89 Months

Productivity: P[i]=KLoC/E[i]

=7.808/20.74

= 0.3761 Per month

- **Proposed Model: E=a_i (KLoC) $^{(bi)}$*Complexity**

KLOC= 7749/1000

= 7.749

Efforts: E[i]= a[i]* (KLoC) $^{(bi)}$ *Complexity

= 2.4* 7.749 $^{(1.05)}$ * 1.25

= 15.45 Man-Month

Development: D[i]= c[i]* E[i] $^{(di)}$

= 2.5*15.45$^{0.38}$

= 7.07 Months

Productivity: P[i]=(KLoC/E[i])

=(7.749/15.45)

= 0.501 Per month

COCOMO				Proposed		
	KLoC=3.327			KLoC=3.327		
	Effort	**Productivity**	**Time**	**Effort**	**Productivity**	**Time**
Organic Mode	**08.47**	**0.39**	**5.63**	**06.35**	**0.52**	**5.04**
Semi detached	**11.52**	**0.28**	**5.38**	**08.64**	**0.38**	**5.31**
Embedded	**10.15**	**0.32**	**5.24**	**08.06**	**0.41**	**4.87**

Table III: Comparison in Low Complexity Project

Table III: In low complexity project we analyze the result with cocomo in that efforts and time is decreases and productivity is increases than Cocomo because of the project is low complexive.

Also In high complexity project we analyze the result with cocomo in that efforts and time is increases and productivity is decreases than Cocomo because of the project is high complexive but we provide the accuracy in estimating cost of that software.

CONCLUSION:

This research presented a modified software reliability growth model that is based on debugging software and detecting faults that might be removed from the software. I implement this reliability growth model along with quality and testability analyzer. Quality and testability analyze the coadaded file from input project and give the suggestion to the users for improving performance. This Reliability Growth Model also calculate the cost of the project that cannot calculated existing model. This model check the developers development skills according to written code files of developer. Also checks efficiency of the developments activities.

REFERENCES:

[1]Yamada, Shigeru, Mitsuru Ohba, and Shunji Osaki. "S-shaped reliability growth modeling for software error detection." IEEE Transactions on reliability 32.5 (1983): 475-484.

[2]Malaiya, Y. K., Li, M. N., Bieman, J. M., & Karcich, R. (2002). Software reliability growth with test coverage. IEEE Transactions on Reliability, 51(4), 420-426.

[3]Pham, H., & Zhang, X. (2003). NHPP software reliability and cost models with testing coverage. European Journal of Operational Research, 145(2), 443-454.

[4]Ledoux, J. (2003). Software reliability modeling. Handbook of Reliability Engineering, 213-234.

[5] Zheng, J. (2009). Predicting software reliability with neural network ensembles. Expert systems with applications, 36(2), 2116-2122.

[6] Martens, A., Koziolek, H., Becker, S., & Reussner, R. (2010, January). Automatically improve software architecture models for performance, reliability, and cost using evolutionary algorithms. In Proceedings of the first

joint WOSP/SIPEW international conference on Performance engineering (pp. 105-116). ACM.

[7] Martens, A., Koziolek, H., Becker, S., &Reussner, R. (2010, January). Automatically improve software architecture models for performance, reliability, and cost using evolutionary algorithms. In Proceedings of the first joint WOSP/SIPEW international conference on Performance engineering (pp. 105-116).ACM.

[8] Zio, E. (2009). Reliability engineering: Old problems and new challenges. Reliability Engineering & System Safety, 94(2), 125-141.

[9]Aljahdali, S., &Sheta, A. F. (2011, April). Predicting the reliability of software systems using fuzzy logic. In Information Technology: New Generations (ITNG), 2011 Eighth International Conference on (pp. 36-40). IEEE.

[10]Lohmor, S., &Sagar, B. B. (2014). Overview: Software Reliability Growth Models. Int. J. Comput. Sci. Inf. Technol.

[11]Pawar, V. E., Kadam, A. K., & Joshi, S. D. (2015). Analysis of Software Reliability using Testing Time and Testing Coverage.International Journal of Advance Research in Computer Science and Management Studies.

[12]Washizaki, H., Honda, K., & Fukazawa, Y. (2015, August). Predicting release time for open source software based on the generalized software reliability model. In Agile Conference (AGILE), 2015 (pp. 76-81). IEEE.

[13]Pawar, V. E., Kadam, A. K., & Joshi, S. D. (2015). Analysis of Software Reliability using Testing Time and Testing Coverage.International Journal of Advance Research in Computer Science and Management Studies.

[14]Mane, M., Joshi, M., Kadam, A., & Joshi, S. D. (2014). Software Reliability and Quality Analyser with Quality Metric Analysis Along With Software Reliability Growth Model.International Journal of Computer Science & Information Technologies, 5(3).

[15]Sabnis, P., & Kadam, A. Software Reliability Growth Model with Bug Cycle and Duplicate Detection Techniques.