ISSN: 2321-7782 (Online)

Volume 1, Issue 5, October 2013

International Journal of Advance Research in Computer Science and Management Studies

Research Paper

Available online at: www.ijarcsms.com

A Study of Empirical Prediction of Defect

Raghvendra Omprakash Singh[1]
Computer Engineering Department
G.M V I T, Mumbai University
Mumbai - India

Rohit S.Kulkarni[2]
Computer Engineering Department
S.E.C.S Gopalpura Sikar
RTU, Kota - India

Aryan Chandrpal Singh[3]
Computer Engineering Department
JCE, Pune University
Pune - India

Abstract: ***Quality of software is dependent on various attributes such as testing, metric and prediction of bugs before deployment which will lead to effective maintenance. Software complexity and bugs again are interrelated. In this paper we are making a comparative study of defect prediction mechanisms. We propose few design ideas for empirical prediction of defect decay. Our research direction will be triggered by the design ideas we are going to propose. We intend to propose a holistic model for correct prediction of defect decay. We also want to perform empirical validation of our model and fine tune it so that its estimates are better than state-of-the-art.***

Keywords: Defect Prediction, defect decay, quality, testing, metrics.

I. INTRODUCTION

Software quality vis-à-vis product success is dependent on testing, code analysis and tackling code based vulnerabilities [19]. Software testing deals with finding errors and metrics collection follows testing [19]. As per ISO 9126[21] factors affecting quality are functionality, efficiency, usability, reliability, maintainability and portability [20]. Kshirasagar Naik et al say, “Efforts to improve quality have centered around the end of the product development cycle by emphasizing the detection and correction of defects” [10]. Kshirasagar Naik et al further add “Software system may be defective due to design issues; certain system states will expose a defect, resulting in the development of faults defined as incorrect signal values or decisions within the system”[10].[14] quotes akiyama for indicating that total number of defects is sum of defects found during testing and found during two months after release.[14] quotes Ferdinand and says defects increase with increase in code segments. As per Stephen G. Eick et al, “Code is decayed if it is more difficult to change than it should be, as reflected by three key responses: (1) COST of the change, which is effectively only the personnel cost for the developers who implement it; (2) INTERVAL to complete the change the calendar/clock time required; and (3) QUALITY of the changed software” [4]. Stephen G. Eick et al further add, "Causes of Code Decay: Inappropriate architecture, Violations of the original design principles, Imprecise requirements, Time pressure, Inadequate change processes, Bad project management"[4]. As per A.Mockus et al, "code decay is the result of previous changes to the software"[2]. Defect prediction indicates predicting the defects in a system and is based on size and complexity metrics [14]. The objective is to design prediction models then empirically validate it by comparing different models using statistical analysis and estimation theory. The focus is on ED^{3}M Model and tries to extract design ideas from its future work [20].

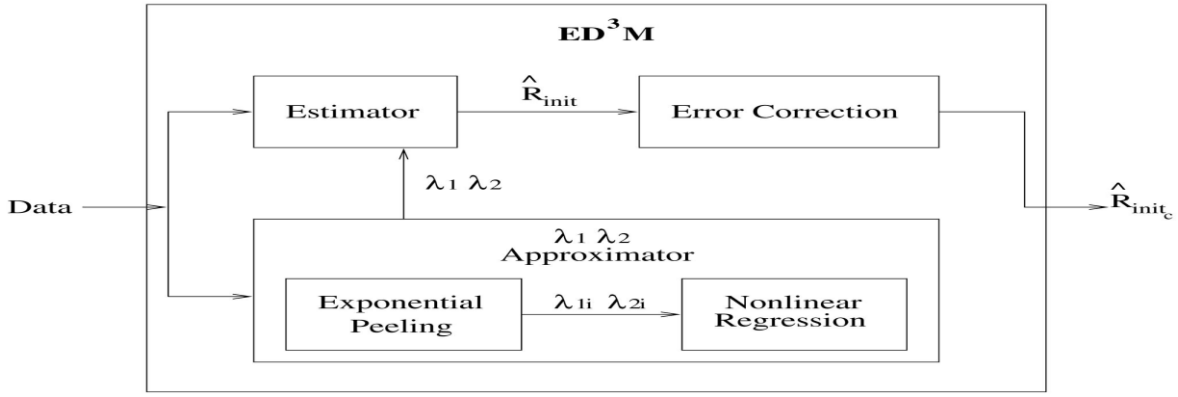

Figure 1. ED3M extracted from (Syed Waseem Haider et al, 2008)

The organization of the paper is as followed: Section 1 introduces the concepts quality, defect, defect decay and defect prediction, Section 2 presents the state-of-the-art about defect prediction techniques. In Section 3, we present our design ideas for defect prediction mechanism. In Section 4, we present the required features for the prediction model which we intend to propose. Finally, Section 5 concludes the paper and discusses possible future works.

II. STATE OF THE ART

Most of the defect prediction schemes use reliability, probability and Bayesian schemes. The approach to defect handling is through defect prediction, failure estimation and defect count and defect density. Statistical models, size and complexity are also used for defect prediction. In [20] "the focus is on characterizing the status of the software testing effort using a single key metric: the estimated number of defects in a software product. The availability of this estimate allows a test manager to improve his planning, monitoring, and controlling activities; this provides a more efficient testing process". Features of ED^{3}M[1]: Estimation of Defects based on Defect Decay Model (ED3M) takes defect count, an almost ubiquitous input, as the only data required to compute the estimates (historical data are not required).Second, the user should not be required to provide any initial values for internal parameters or expert knowledge; this results in a fully automated approach. Third, the technique should be flexible; it should be able to produce estimates based on defect data reported in execution time or calendar time. The only input is the defect data; the ED3M approach is fully automated. Issues mentioned in ED^{3}M: there are defect prediction issues that are not addressed by it. For example, system test managers would benefit from obtaining a prediction of the defects to be found in ST well before the testing begins, ideally in the requirements or design phase. This could be used to improve the plan for developing the test cases. The ED^{3}M approach, which requires test defect data as the input, cannot be used for this. Alternate approaches which rely on different input data (e.g., historical project data and expert knowledge) could be selected to accomplish this. However, in general, these data are not available at most companies. A second issue is that test managers may prefer to obtain the predictions for the number of defects on a feature-by-feature basis, rather than for the whole system. Although the ED3M approach could be used for this, the number of sample points for each feature may be too small to allow for accurate predictions. As before, additional information could be used to achieve such estimations, but this is beyond the scope of this paper. Third, the performance of the ED^{3}M approach is affected when the data diverge from the underlying assumption of an exponential decay behavior. However, the results indicate the estimations are still useful under these conditions. Our approach takes guidance from this previous work, but is notably different by suggesting new prediction models and by using an information theoretic approach to measure the effectiveness of such models.[1]

Overview of ED^{3}M: "The only input to the ED^{3}M model is the set of data observations; no additional a priori knowledge is required. The data contains the number of defects removed from the software under test; the data may be sampled using any

[1] Extracted from Syed Waseem Haider, Joa˜o W. Cangussu, Kendra M.L. Cooper and Ram Dantu,"Estimation of Defects Based on Defect Decay Model: ED3M",I EEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 34, NO. 3, MAY/JUNE 2008

given fixed period of time (e.g., daily, weekly, bi-weekly, etc.). The output of the ED^{3}M model is an estimate of the total number of defects in the software, Rinitc . As additional data become available, the estimate may be recalculated. The ED3M model has three main components: the Estimator, the λ1 λ2 Approximator, and the Error Correctionblock (refer to Fig. 1). The Estimator component is responsible for calculating an estimate of the total number of defects in the product. It requires the data observations and initial values for two constants, λ1 and λ2, as inputs. The two constants are generated by the λ1 λ2 Approximator. The estimator is based on an existing model for the software test process called the Defect Decay Model (DDM); this component calculates and outputs the estimate, ^Rinit. The λ1 λ2 Approximator component is responsible for finding two constants, λ1 and λ2; these are the rate and scale parameters for the defect curve over time. The inputs are the data observations. Two existing techniques are applied in this component: exponential peeling and nonlinear regression. The approximation approach is also based on the DDM for the software test process. This component calculates and outputs the values for λ1 and λ2.The Error Correction component is responsible for improving the estimate and, consequently, improving the convergence time for the ED^{3}M model. The input is the estimate ^Rinit calculated by the Estimator component. The algorithm in this component calculates a mean growth factor using a history of previous estimates, where each growth factor value is the ratio of one estimate over a previous estimate. The mean growth factor is used to correct the current estimate; the corrected value is the output, ^Rinitc .The DDM [9] is a mathematical model capturing the dominant behavior of the system test phase. It is given by(1), where R(t) is the number of remaining errors at time t, R(t) is the error reduction velocity, and R(t) is the rate of change of error reduction velocity. The model parameters are viz., w_f (the work force), s_c (the software complexity), γ (the overall quality of the test phase), and ζ and ξ are constants.

$$-\frac{\zeta\, w_f}{s_c^b}R(t) - \xi\frac{1}{\gamma}\dot{R}(t) = s_c\ddot{R}(t). \tag{1}$$

A solution of (1) where Rinit is the initial number of defects present in the software at time t ¼ 0 and the error reduction velocity R(t) is zero at t ¼ 0 has a double exponential form given by

$$R(t) = R_{init}\left(\frac{\lambda_2}{\lambda_2-\lambda_1}e^{-\lambda_1 t} - \frac{\lambda_1}{\lambda_2-\lambda_1}e^{-\lambda_2 t}\right). \tag{2}$$

".[1]

As quoted by Syed et al: "Padberg [6] proposed an algorithm to find maximum likelihood estimates for Hypergeometric Distribution Software Reliability Growth Model (HGDM) to predict the number of defects initially present in the software. Note that some of the symbols used to explain Padberg's approach may be the same as used earlier in this paper, but they represent different concepts here; the new definitions are provided. Padberg has shown that the growth quotient Q(m) of the likelihood function L(m) when greater than 1 indicates that the likelihood function is indeed increasing and provides maximum likelihood estimates:

$$Q(m) = \frac{L(m)}{L(m-1)} = \frac{(m-w_1)\ldots(m-w_n)}{m^{n-1}\cdot(m-c_n)}. \tag{3}$$

."

As quoted in Syed et al: "The Gompertz curve model given by (4) is based on the S-shaped behavior. In general, a nonlinear regression using the Gauss-Newton method is used to estimate the three parameters Rinit, b, and k, which characterizes the Gompertz curve.

$$G(t,\Theta) = R_{init}e^{-be^{-kt}} \tag{4}$$

The use of SRGM based on the Gompertz curve can lead to accurate estimations of the number of defects. However, the results are heavily dependent on the initial values of the parameters used in the estimation. In real, ongoing projects where the actual number of defects is not known (i.e., in the absence of a reference point), it is very difficult to find the right initial values. In contrast, ED3M as pointed out is a turn-key solution that does not require any initial values."

Testing Type	Defects found per hour
Regular use	0.210
Black-box	0.282
White-box	0.322
Reading/Inspections	1.057

Table 1. Defects found per testing approach extracted from (Grady 1992)

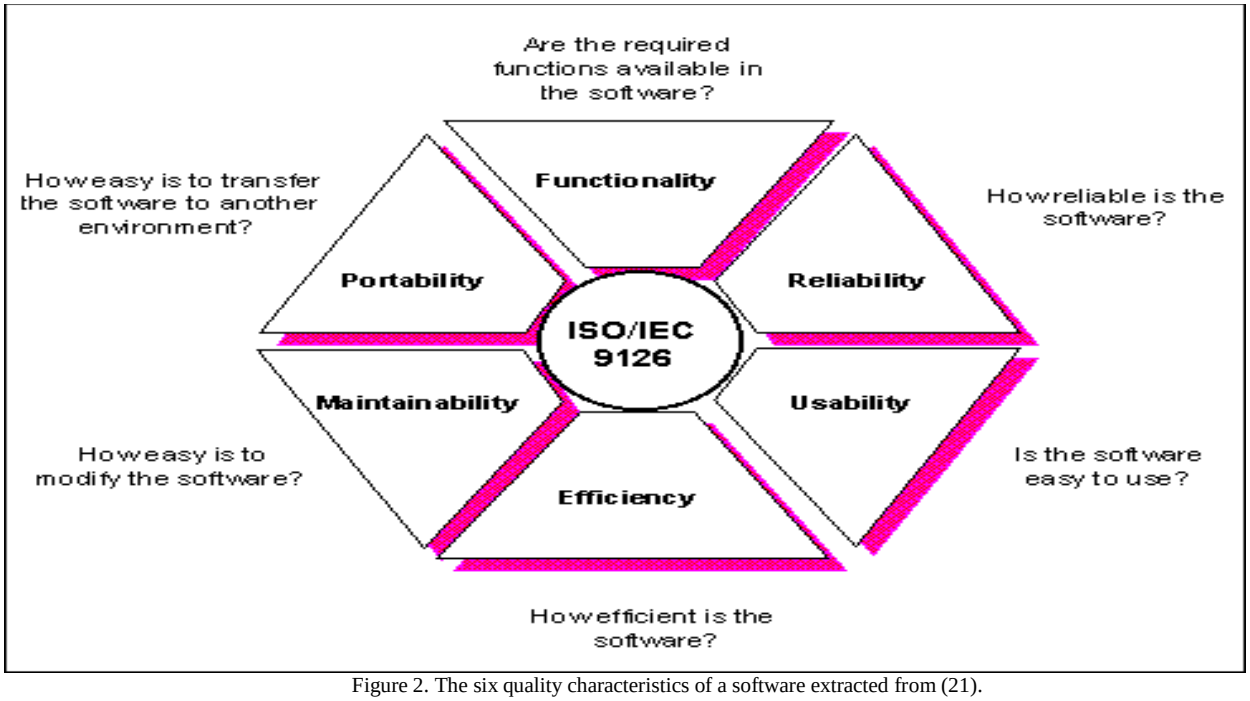

Figure 2. The six quality characteristics of a software extracted from (21).

III. Defect Prediction Design Ideas

Correctly predicting defects is still open-ended. Our objective is to improvise on ED3M model and show higher convergence with lower error rate. There are variations in convergence rate of ED3M which we want to stabilize. We also want to evolve techniques which exhaustively predict defects at the defect saturation phase. Also during initialization phase convergence rate improvement methodology we intend to design. We would like to propose a sturdy model with a balanced behavior irrespective of variation in parameters.

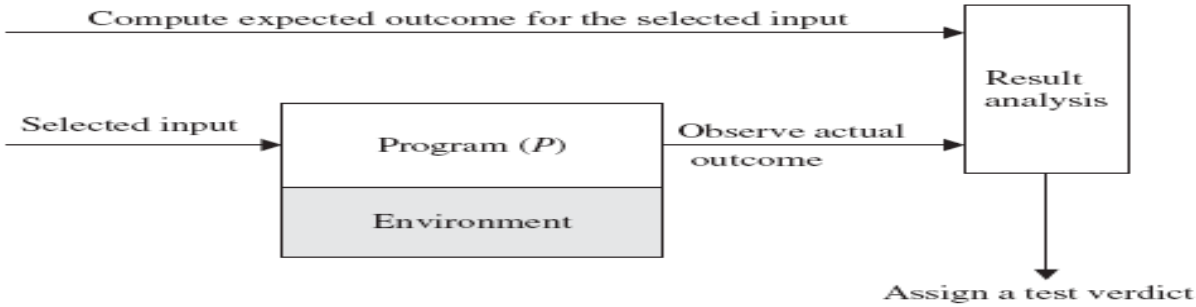

Figure 3. Different activities in Program testing extracted from (10).

Case Study	Convergence Time		System Test Accuracy
	±20% Tolerance	±30% Tolerance	
# 1	17%	17%	2.2%
# 2	25%	16%	5.8%
# 3	36%	30%	6%
# 4	41.3%	38%	7.2%
# 5	34%	34%	6%

Table 2. ED^{3}M Applied on case studies extracted from (Syed Waseem Haider et al, 2008)

IV. Prediction Model Features

ED^{3}M model uses estimation theory, estimates defects, predicts total number of defects and approximates defect decay through error correction model.

Our goal is to use this predictive model as base model and improvise upon it. In the basic DDM model work force w_f as well as complexity s_c has variations with noise and error. ED^{3}M model treated it as a constant. We would like to add increment/decrement mobility to the constant with noise and error. Other quality factors such as usability can also be considered for addition.

V. Conclusion

Defect decay prediction model along with Bayesian models are visible efforts towards defect prediction. In this paper we discussed theory behind defect prediction as a product quality component. We resolved to use ED3M as a base model to propose our prediction model. We presented some design ideas and intended features for our prediction model. We hope our design ideas and features section in this paper will become a road map for our research journey.

References

1. Allen, J. F. Using Entropy for Evaluating and Comparing Probability Distributions, available at: http://www.cs.rochester.edu/u/james/CSC248/Lec6.pdf
2. A. Mockus, S. G. Eick, T. L. Graves, and A. F. Karr, "New roles for change management data in software engineering", Technical Report, National Institute of Statistical Sciences, 1999.
3. Basili, V. R., and Perricone, B. Software errors and complexity: An empirical investigation. Communications of the ACM, 27(1):42 – 52, 1984.
4. Eick, S. G., Graves, T. L., Karr, A. F., Marron, J.S., and Mockus, A. Does Code Decay? Assessing the Evidence from Change Management Data. IEEE Trans. on Software Engineering, 27(1):1–12, 2001.
5. Eick, S.G., Graves, T.L., Karr, A.F., Mockus, A., Schuster, P. Visualizing Software Changes, IEEE Trans. on Software Engineering, vol. 28, no. 4, pp. 396-412, April, 2002.
6. F. Padberg, "Maximum Likelihood Estimates for the Hypergeometric Software Reliability Model," Int'l J. Reliability, Quality, and Safety Eng., July 2002.
7. Graves, T. L., Karr, A. F., Marron, J. S. and Siy, H. P. Predicting fault incidence using software change history. IEEE Trans. on Software Engineering, 26(7):653–661, 2000.
8. J.D. Musa, A. Iannino, and K. Okumoto, Software ReliabilityMeasurement, Prediction, Application. McGraw-Hill, 1987.
9. J.W. Cangussu, R.A. DeCarlo, and A.P. Mathur, "A Formal Model for the Software Test Process," IEEE Trans. Software Eng., vol. 28,no. 8, pp. 782-796, Aug. 2002.

10. Kshirasagar Naik, Priyadarshi Tripathy,"Software Testing and Quality Assurance, Theory and Practice",WILEY,A JOHNWILEY & SONS, INC., PUBLICATION.

11. Khoshgoftaar, T. M., Allen, E. B., Jones, W. D., and Hudepohl, J. P. Data Mining for Predictors of Software Quality. International Journal of Software Engineering and Knowledge Engineering, 9(5), 1999.

12. Manning, C. and Schütze, H. Foundations of Statistical Natural Language Processing, MIT Press. Cambridge, MA: May 1999.

13. Mockus, A. and Votta, L. G. Identifying reasons for software change using historic databases. In International Conference on Software Maintenance, pages 120-130, San Jose, California, October 11-14 2000

14. Norman Fenton,Martin Neil, "A Critique of Software Defect Prediction Models" ,Centre for Software Reliability.

15. Ostrand, T. J., Weyuker, E. J., Bell, R. M. Predicting the Location and Number of Faults in Large Software Systems. IEEE Trans. Software Eng. 31(4): 340-355 (2005)

16. Pareto Law: http://www.it-cortex.com/Pareto_law.htm

17. Perry, D. E. and Evangelist, W. M. An Empirical Study of Software Interface Faults — An Update. In Proceedings of the 20th Annual Hawaii International Conference on Systems Sciences, pages 113–136, Hawaii, USA, January 1987.

18. Perry, D. E. and Steig, C.S. Software Faults in Evolving a Large, Real-Time System: a Case Study'. In Proceedings of the 4th European Software Engineering Conference, Garmisch, Germany, September 1993.

19. Rajesh Kulkarni, P. Padmanabham & M. S. Namose, "Improving Software Quality Attributes of PS using Stylecop",Global Journal of Computer Science and Technology Volume XIII Issue VIII Version I,2013

20. Syed Waseem Haider, Joa˜o W. Cangussu, Kendra M.L. Cooper and Ram Dantu,"Estimation of Defects Based on Defect Decay Model: ED3M",IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 34, NO. 3, MAY/JUNE 2008

21. http://www.issco.unige.ch/en/research/projects/ewg95//node1.html

AUTHOR PROFILE

RAGHVENDRA OMPRAKASH SINGH, received his bachelor's degree in Computer Science & Engineering from Solapur University. He is currently Persuing Mtech in Computer Engineering from Rajasthan Technical University and also working as an Assistant Professor with the Department of Computer Engineering, G.M Vedak Institute of technology-Tala, Mumbai University India. His research interests mainly focused on Software Testing, Defect Decay Models, and Quality.

ROHIT SHASHIKIRAN KULKARNI, recived his bachelor's degree in Computer Science & Engineering from Solapur university, He is currently doing Mtech in Computer Science Engineering from Rajasthan Technical University. He was Lecturer at BVIT Navi Mumbai.

ARYAN CHANDRAPAL SINGH, received his bachelor's degree in Computer Science and Engineering from Gautam Budh Technical University, He is currently doing Master in technology in Computer Science and Lectureship in Jai Hind College of Engineering – Narayangaon, Pune University.

