ISSN 2319 – 7757 EISSN 2319 –7765

discovery

Indian Journal of Engineering

Evaluation for Defective Density in All the Right Places

Chandrakanth G Pujari[1*], Seetharam K[2]

1. Associate Professor, MCA Dept. Dr.AIT, Bangalore-560056, India & Research Scholar, Sathyabama University Chennai, Email: chandrapujar12@gmail.com
2. Professor, Computer Science and Engineering Dept., RGIT Bangalore-560032, India, E-mail: amoghks@rediffmail.com

[*]Corresponding author: Associate Professor, MCA Dept. Dr.AIT, Bangalore-560056, India & Research Scholar, Sathyabama University Chennai, Email: chandrapujar12@gmail.com

Publication History
Received: 21 September 2014
Accepted: 3 October 2014
Published: 12 November 2014

Citation
Chandrakanth G Pujari, Seetharam K. Evaluation for Defective Density in All the Right Places. *Indian Journal of Engineering*, 2014, 11(26), 30-37

ABSTRACT

Investigating the use of a binomial distribution to predict which files in a very large software organizational software system are most likely to contain many defects in the software projects. A new empirical study is described whose subject is an automated software system. Not only is this system's functionality substantiality different from that of the earlier systems we studied, it also uses a significantly different software development process. Alternate schedule and releases as both of the earlier software systems did, this software system has what are software system referred to as "continuous software projects produces." Discusses the help of the binomial distribution, as well as a simple software projects lines-of-code based model, to make predictions for this software system and discuss the differences observed from the earlier research. Despite the different software project development procedure, the very good version of the software development project prediction model was able to identify, over the lifetime of the project, 20% of the system's files that consists, on arithmetic mean, nearly three quarters of the software defects that were detected in the software systems. In this paper an important characteristic of software system is that it must produce functionally or logically accurate calculated results within a predefined time period. Hence, controlling the software defects is very critical for any software system. One approach for realizing this is by predicting the software defect density and taking suitable actions if the calculated value is higher than the set target. A lot of software defect prediction models based on several methods have been proposed in the past. In this paper we used the Bayesian approach to develop the prediction model. Bayesian modeling is very useful when a very complex relationship between the variables exists and when all of the predictor variables are discontinuous. It is very easy to do what-if analysis or sensitivity analysis with Bayesian models. A brief illustration of the aforementioned approach is presented in this paper.

Keywords: Bayesian model, defect density, software system, Binomial model, empirical study

1. INTRODUCTION

In industry, post-release defect density of a software system cannot be measured until the system has been put into production and has been used extensively by end users (Adams, 1984). The defect density of a software system is calculated by measuring the number of defects divided by the size of the system, using a size evaluates such as LOC (V.R.Basili and B.T.Perricone, 1984). Actual post-release defect density information becomes available too late in the software lifecycle to affordably guide corrective actions to software quality (G.Denaro and M.Pezze, 2002). Correcting software defects is significantly more expensive when the defects are discovered by an end user compared with earlier in the development process (M.Pighin and A.Marzona, 2003). Because of this increasing price of correcting software defects, software developers can profitable from early calculation of the software defect density of their software product. If software developers can be presented with this software defect density information during the software development process in the software environment where they are creating the software system, provide more corrective action can be taken to rectify software defect density concerns as they appear (Fenton. N.E, S.L. Pfleeger, 1996). In industry, information on defect density of a product tends to become available too late in the software development process to affordably guide corrective actions. Since substantial resources are expanded in efforts to make large industrial software systems highly dependable, we have been developing techniques to accurately predict which files in a software system are most likely to be problematic in the next release. That information should help alert software developers to files that may need to be rewritten or to which particular care should be paid and help testers to prioritize and focus their efforts (Jenson.F.V, 2009). In this paper we introduced a binomial model that we used to accurately predict which files are likely to contain the largest numbers of defects (Dolbec, J. and Shepard, T., 1995). These predictions were based on file characteristics that could be objectively assessed, including the size of the file in terms of the number of lines of code (LOC), whether this was the first release in which the file appeared, whether files that occurred in earlier released had been changed or remained unchanged from the previous release, how many previous releases the file occurred in, how many defects were detected in the file during the previous release, and the programming language in which the file was written (T.M.Khoshgoftaar, E.B.Allen, J.Deng, 2002). We selected these characteristics are most relevant ones.

2. EVALUATION OF DEFECT DENSITY IN SOFTWARE PROJECTS

Table 1 provides information on the software system for each of the 29 calendar months that were tracked. A total of 1928 files entered the system at some point over the 29 months, 1926 of which remained through the last month (one file was dropped after each of months 12 and 20). Over the 29 months studied, a total of 1482 defects were detected, mostly during software testing with 329,070 lines in its latest version, the software system is comparable in size to the systems considered. The rightmost column of Table 1 shows the monthly defect density (defects detected per thousand LOC) of the system. We adjust defect densities to correspond to rates per quarter (three months), that is, defect density = defects per KLOC-quarter. For example, the defect density shown for month 2 is (3*116)/59.371 = 5.86 Binomial model is an extension of linear regression model to handle outcomes that are nonnegative integers-such as the number of defects in a file during a specified period, in contrast to Poisson regression, which assumes that defects occur “at random” at a rate explained by a set of predictor variables, the binomial model allows for some degree of additional variability in fault counts that is not explained by any of the available predictor variables, technically referred to as over dispersion. Because we want to predict the number of defects discovered for an individual file in a specific month, the unit of analysis is the combination of a file and a month. Consequently, a single file may contribute many observations, corresponding to the number of months it is in the system. Let observation I refer to a specific combination of file and month. We let yi denote the number of defects observed and xi be the vector of file characteristics for that month. Like Poisson regression, binomial regression models the logarithm of the expected number of defects as a linear combination of the explanatory variables. That is, the model assumes that the expected number of defects varies in a multiplicative way with file characteristics, rather than in an additive relationship.

The binomial regression model specifies that yi has a Poisson distribution with mean $\lambda i = \gamma i * e^{\beta' xi.}$ A random variable γi, drawn from a gamma distribution with mean 1 and unknown variance σ2>=0, is included to account for the additional dispersion (variation in the number of defects) observed in the data. Its variance σ2, known as the dispersion parameter, measures the magnitude of the unexplained concentration of defects.

The regression coefficients β and the dispersion parameter σ2 are typically estimated by maximum likelihood. All estimation was performed using Version 9.1 of SAS user guide. Once a model has been estimated, it is simple to compute the predicted numbers of defects for files in subsequent months, as long as the same set of explanatory variables is measured for all files in the new month. These predictions may be used to prioritize files in terms of their

Chandrakanth and Seetharam,
Evaluation for Defective Density in All the Right Places,
Indian journal of engineering, 2014, 11(26), 30-37,
http://www.discovery.org.in/ije.htm

expected number of defects, which can then be used for purposes of testing. In this paper, the phrase actual defect always refers to the defects that have been detected. Obviously, a system may contain undetected defects, but there is no way for us to be aware of them. In this paper we continue our investigation using a substantially different type of system with very different characteristics. This is an automated software system which is used by many companies to provide customer service while limiting reliance on human operators. The most significant differentiating characteristics of this system is the lack of a formal release schedule, which made the straightforward application models.

3. CODE REVIEW FOR DEFECT DENSITY

Software development systems are determining with tightly coupled hardware and software that are designed to perform a dedicated function. The software development system is referred to as real time duration when it is expected to perform a function in response to external events, and the response is time critical. Responding to external events includes identifying when an event occurs, performing the required processing as a result of the event, and giving out the necessary results within a given time constraint. The most important characteristics of real-time software development systems are that they must produce correct calculated results, called logical or functional correctness, and that these calculations must conclude within a predefined time period, called timing correctness. Along with cost and time, the quality or reliability of the software is also very serious (Jalote, 2000).

The quality or reliability of software is often evaluated in terms of software defect density. The software defect density is defined as the number of software defects per unit size of the software (Fenton and Pfleeger, 1996). This research was undertaken to develop a prediction model for final software defect density for software development system, namely "software project development." A software component that is responsible for managing the low-level input and output operations of a software tools. In other words, a software tools is a piece of software that acts like a "liaison" between the system software and the application software.

Many researchers have attempted to identify software properties that correlate with fault-prone code for many years. Some of this work is reported in. Attempts to predict the probability, the locations or the quantity of future faults in code are more recent, and include as well as our research described in. Work by (Khoshgofthaar, Allen, and Deng 2002) presents a method to construct a binary decision tree to classify modules of a system as fault-prone or not fault-prone. The tree's decision nodes are based on 24 static software metrics and four execution time metrics. The approach is demonstrated with four releases of a large legacy telecommunications system. The regression-tree construction algorithm of the S-Plus statistics package is applied to training data from the first release, producing the tree that is then evaluated on data of the next releases.

A large number of software defect prediction models have been proposed in the past. The earlier works (Akiyama 1971; Ferdinand 1974; Halstead 1975; Lipow 1982; Gaffney 1984) were linear, polynomial, or nonlinear regression models using size as the predictor variable. Later on, prediction models based on McCabe complexity (Linkman, Kitchenham, and Pickard 1990) metrics derived from design document (Alberg and Ohlsson 1996), function points (Jones 1991), and so on were proposed. Multivariate techniques like principal component analysis (Niel 1992) and factor analysis (Khoshgoftaar and Munson 1990) were also tried in the past. Recently, many models based on advanced tools like logistic regression and Boolean descriminant function (Schneidewind 2001), classification tree (Koru and Liu 2005), Bayesian belief networks (Fenton et al. 2007), neural networks, data mining techniques (Menzies 2007), genetic programming (Liu forthcoming), and so on were proposed. In this paper we discuss the Bayesian approach for developing prediction models (Table 2).

4. BAYESIAN METHOD

The Bayesian method provides a very simple and efficient approach for developing prediction models. It computes the probability of output parameters taking different values given the values of input or predictor parameter. The Bayesian models the dependencies between all of the parameters in the software data. The software data dependencies can also be used to without any firm evidence about the causalities that might cause them. More details on Bayesian method can be found in (Jenson, 2009) and (Frazer, 2010). The major advantages of using Bayesian methods are the following:

The very complex relationship between the parameters can be easily modeled. The parameters can be continuous, discrete, or categorical and even subjective based on expert opinion. It is easy to perform what-if and sensitivity analysis. The most important part of Bayesian networks model is the Bayes theorem:

$$\text{Probability } P(M/N) = [P(N/M)*P(M)] / P(N) \quad (1)$$

Chandrakanth and Seetharam,
Evaluation for Defective Density in All the Right Places,
Indian journal of engineering, 2014, 11(26), 30-37,
http://www.discovery.org.in/ije.htm

This theorem computes the probability of M given the value of N. A simple example is that this technique can be used to compute the probability that the software defect density (y or output parameter) of a code review procedure will lie in an interval given the values of predictor parameters like review session time and code review coverage (X). Consider the code review information given in table 2. Using data in table 2, how the Bayesian model predicts the software defect density of a code review procedure for a given observation X is demonstrated next,

Where X: Review duration = 2 to 3 hrs and code coverage medium (MD)

The first step is to compute the probability of the response parameter taking different possible values. Since there are only two values in table2, for the response parameter software defect density (< 2.5 and >= 2.5), the P(software defect density < 2.5/X) and P(software defect density >= 2.5/X) can be calculated as follows:

P (software defect density < 2.5/X) = P(X / software defect density < 2.5 / X) * P(software defect density < 2.5) / P(X). (2)

P (software defect density >= 2.5 / X) = P(X / software defect density >= 2.5/X) * P(software defect density >= 2.5) / P(X), (3)

Where, probability P (software defect density < 2.5) is the ratio of the number of cases with software defect density < 2.5 to the total number of data points.

P (software defect density < 2.5) = 11/29 = 0.379 (4)

Similarly,

P (software defect density >= 2.5) = 18/29 = 0.621 (5)

The next step is to calculate the P(X/software defect density <2.5) and P(X/software defect density >= 2.5).

The P(X/software defect density < 2.5) is calculated as the product of conditional probability for each value of X:

P(X/software defect density < 2.5) = P (review duration = 2 to 3 hrs/ software defect density < 2.5) * P(code coverage = medium/software defect density < 2.5) (6)

Where P (review duration = 2 to 3 hrs/defect density < 2.5) is computed as the ratio of the number of cases with a review duration of two to three hrs when defect density < 2.5 to the total number of cases with defect density < 2.5

P (review duration = 2 to 3hrs / software defect density < 2.5) = 11/11 = 1 (7)

P (code coverage = medium /software defect density < 2.5) = 2/11 = 0.182 (8)

Therefore, using equation (6), (7), and (8)
P(X/software defect density < 2.5) = 1 * 0.182 = 0.182 (9)

Using equations (4) and (9)
P(X/software defect density >= 2.5) * P (software defect density >= 2.5) = 0.379 * 0.182 = 0.069 (10)

Similarly, P(X/software defect density >= 2.5) * P(software defect density >= 2.5) is calculated as follows

P(X/software defect density >= 2.5) = P (review duration = 2 to 3hrs / software defect density >= 2.5) * P (code coverage = medium/software defect density >= 2.5) (11)

Where P (review duration = 2 to 3hrs/software defect density >= 2.5) = 1/18 = 0.055555 (12)
and
P (code coverage = medium / software defect density >= 2.5) = 14/18 = 0.777 (13)
Using (11), (12), (13)

P(X/software defect density >= 2.5) = 0.055555 * 0.7777 = 0.0432 (14)
Therefore, using equations (5) and (14)

P(X/software defect density >= 2.5) * P(software defect density >= 2.5) = 0.0432 * 0.621 = 0.0268 (15)
Finally, P (software defect density < 2.5 / X) and P(software defect density >= 2.5/X)
Can be calculated using (2) and (3)

P (software defect density <2.5/X) = 0.1/(0.1 + 0.0268) = 0.1/0.1268 = 0.789 = 78.9% (16)

P(software defect density >= 2.5 /X) = 0.0268 / (0.1 + 0.0268) = 0.0268/0.1268 = 0.211 = 21.1% (17)

Hence, the final result is that there is a 78.9% chance that software defect density >= 2.5 and a 21.1 % chance that software defect density < 2.5 for the given X(X = Review duration = 2 to 3hrs and code coverage = medium).

The calculation of these probabilities is easy when there are only one or two predictors or independent parameters. When there are many predictor parameters with multiple values, the computation of probabilities becomes very complicated since a large number of calculations are required.

5. CONCLUSION

Due to the very highest costs of attaching software defects once a software product has reached the field, any data or information that can be provided to software developer's in software process and can give an indication of software defect density is invaluable. If most corrective actions can be taken previously in the software development life cycle to isolate and repair software defects, overall software maintenance costs can decrease. The computation of software defects or software defect density is very serious for any software development. Many requests, varying from linear regression model to neural networks models, have been used in the past to software development defect prediction models. In this paper discussed the Bayesian methods for developing prediction models for the software company. This research was carried out in software development. Because the outputs are very encouraging, planning to undertake similar researches in different domains as part of our consultancy assignments with software development organizations.

ACKNOWLEDGEMENTS

The authors wish to express their gratitude to Dr.AIT Management for all supports. We also thank for software practitioners who gave their experiences and time generously.

REFERENCES

1. Akiyama F, An example of software system debugging. Information Processing 71, 1971, Pp. 353-379
2. A.Mockus and D.M.Weiss. "Predicting Risk of Software Changes", Bell Labs Technical Journal, April-June 2000, Pp.169-180
3. Alberg, H., and N. Ohlsson, "Predicting error-prone software modules in telephone switching", IEEE Transactions on Software Engineering 22", no.12, 1996, Pp.886-894
4. Dolbec, J. and Shepard, T., A Component Based Software Reliability Model, Conference of the Centre for Advanced Studies on C, 1995
5. E.N. Adams. Optimizing Preventive Service of Software Products. IBM J.Res. Develop. Vol. 28. No 1. Jan 1984, pp. 2-14
6. Fenton. N.E, and S.L.Pfleeger, Software metrics. A rigorous and practical, 2nd edition. New York: International Thomson Computer Press. 1996.
7. Frazer, D. A. S. Bayesian inference: An approach to statistical inference. Wiley Interdisciplinary Reviews: Computational Statistics 2", 2010, no. 3: 487-496
8. Ferdinand.A.E, "A theory of system complexity", International Journal of General Systems 1, 1974, Pp. 19-33
9. G.Denaro and M.Pezze. An Empirical Evaluation of fault. Proneness Models. Proc. International Conf on software Engineering (ICSE2002), Miami. USA, May 2002
10. Gaffney, J. R. "Estimating the number of faults in code", IEEE Transactions on Software Engineering SE-10,1984 no. 4, Pp.324-334
11. Halstead, M. H. Elements of Software Science. North-Holland: Elsevier 1975
12. Jones, C, Applied software measurement. New York: McGraw Hill 1991
13. J.C.Munson and T.M.khoshgoftaar. "The Detection of Fault-Prone Programs", IEEE Trans. On Software Engineering. Vol 18, No 5, May 1992, pp. 423-433
14. Jenson. F.V, Bayesian networks. Wiley Interdisciplinary Reviews: Computational Statistics 1, 2009, no.3: 307- 315
15. Jalote.P. CMM in practice for executing software projects at Infosys. Upper Saddle River, NJ: Addition-Wesley Longman Inc. 2000
16. K.H.Moller and D.J.Paulish. "An Empirical Investigation of Software Fault Distribution" Proc. IEEE First International Software Metrics Symposium, Baltimore, Md., May 21-22, 1993, pp. 82-90
17. Koru, A. G., and H. Liu., "Building effective defect-prediction models in practice", IEEE Software 22, 2005, no.6, pp. 23-29

18. L.Guo, Y.Ma, B.Cukic, H.Singh, "Robust Prediction of Fault-Proneness by Random Forests", Proc. ISSRE 2004, Saint-Malo, France, Nov. 2004
19. L.Hatton, "Reexamining the Fault Density-Component Size Connection", IEEE Software, March/April 1997, pp. 89-97
20. Lipow, M. "Number of faults per line of code", IEEE Transactions on Software Engineering SE-8,1982, no. 4:437-439
21. Linkman, S. J., B. A. Kitchenham, and L.M. Pickard. "An evaluation of some design metrics", Software Engineering Journal, 1990, no. 5 pp. 50-58.
22. Menzies, T. "Data mining static code attributes to learn defect predictors." ,IEEE Transactions on Software Engineering 33, 2007, no. 1:1-13
23. M. Pighin and A.Marzona. "An Empirical Analysis of Fault Persistence through Software Releases", Proc. IEEE/ACM ISESE 2003, pp. 206-212
24. N.E.Fenton and N.ohisson. "Quantitative Analysis of Faults and Failures in a Complex Software System", IEEE Trans. On Software Engineering. Vol 26, No. 8, Aug 2000, pp. 797-814
25. N.ohisson and Halberd. "Predicting Fault-Prone Software Modules in Telephone Switches", IEEE Trans. On Software Engineering, Vol 22, No. 12, December 1996, pp. 886-894
26. Niel, M. D.," Multivariate assessment of software products", Journal of Software Testing, Verification and Reliability, 1992, no.4, pp. 17-37
27. P.McCullagh and J.A. Nelder. Generalized Linear Models, Second Edition, Chapman and Hall, London, 1989
28. Schneidewind, N.F. "Investigation of logistic regression as a discriminant of software quality", in Proceedings of IEEE 7TH International Software Metrics Symposium, 4-6 April, London, 2001, pp. 328-333
29. S.G.Eick, T.L.Graves, A.F.Karr, J.S.Marron. A.Mockus." Does Code Decay? Assessing the Evidence from Change Management Data", IEEE Trans. On Software Engineering, Jan 2001, Vol 27. No. 1, pp. 1-12
30. SAS Institute Inc. SAS/STAT 9.1 User's Guide, SAS Institute, Cary, NC, 2004
31. T.L.Graves, A.F.Karr, J.S.Marron, and H.Siy.," Predicting Fault Incidence Using Software Change History", IEEE Trans. On Software Engineering, Vol 26, July 2000, No. 7, pp. 653-661
32. T.M.khoshgoftaar, E.B.Allen, K.S.Kalaichelvan, N.Goel. "Early Quality Prediction: A Case Study in Telecommunications", IEEE Software, Jan 1996, pp. 65-71
33. T.M.khoshgoftaar, E.B.Allen, J.Deng. "Using Regression Trees to Classify Fault-Prone Software Modules", IEEE Trans. On Reliability. Vol 51, No. 4, Dec 2002, pp. 455-462
34. V.R. Basili and B.T. Perricone. Software Errors and Complexity: An Empirical Investigation. Communications of the ACM, Vol. 27 No 1. Jan 1984, pp. 42-52

Table 1
Software defects and defect density

Month	Number of Project Files	Project Lines of Code	Average Lines of Code	Changes Made	Defects Detected	Defect Density
1	61	15,706	257.5	19	3	0.57
2	325	59,371	182.7	257	116	5.86
3	433	80,619	186.2	237	29	1.08
4	611	107,131	175.3	310	119	3.33
5	758	131,742	173.8	398	119	2.71
6	873	151,232	173.2	247	100	1.98
7	959	161,892	168.8	327	82	1.52
8	1060	174,267	164.4	302	91	1.57
9	1203	202,700	168.5	185	32	0.47
10	1281	210,123	164.0	289	121	1.73
11	1312	217,451	165.7	159	97	1.34
12	1334	229,152	171.8	177	89	1.17
13	1404	241,127	171.7	252	81	1.01
14	1468	252,414	171.9	174	71	0.84
15	1597	265,380	166.2	174	40	0.45
16	1756	297,325	169.3	271	62	0.63
17	1788	301,189	168.5	160	23	0.23
18	1807	304,435	168.5	54	9	0.09
19	1809	304,702	168.5	26	22	0.22
20	1821	306,019	168.0	36	4	0.04
21	1825	306,688	168.0	107	52	0.51
22	1825	306,765	168.1	41	12	0.12
23	1846	313,276	169.7	129	29	0.28
24	1866	316,422	169.6	89	32	0.30
25	1887	319,854	169.5	85	25	0.28
26	1900	321,782	169.4	96	11	0.10
27	1906	324,533	170.3	56	2	0.02
28	1921	326,518	170.0	42	7	0.06
29	1926	329,070	170.9	71	2	0.02

Table 2
Code review for software data

Software project code review duration	Coverage	Software defect density
>= 1hour and <= 2hours	Medium	>= 2.5
>= 1hour and <= 2hours	Medium	>= 2.5
>= 1hour and <= 2hours	Medium	>= 2.5
>2hours and <=3hours	High	< 2.5
>= 1hour and <= 2hours	Medium	>=2.5
>2hours and <=3hours	High	<2.5
>= 1hour and <= 2hours	Medium	>=2.5
>2hours and <=3hours	High	<2.5
>= 1hour and <= 2hours	High	>=2.5
>= 1hour and <= 2hours	High	>=2.5
>= 1hour and <= 2hours	Medium	>=2.5
>= 1hour and <= 2hours	Medium	>=2.5
>2hours and <=3hours	High	<2.5
>= 1hour and <= 2hours	Medium	>=2.5
>= 1hour and <= 2hours	High	>=2.5
>2hours and <=3hours	High	<2.5
>2hours and <=3hours	Medium	<2.5

>= 1hour and <= 2hours	Medium	>=2.5
>2hours and <=3hours	High	<2.5
>2hours and <=3hours	Medium	>=2.5
>= 1hour and <= 2hours	Medium	>=2.5
>= 1hour and <= 2hours	Medium	>=2.5
>2hours and <=3hours	High	<2.5
>= 1hour and <= 2hours	Medium	>=2.5
>= 1hour and <= 2hours	High	>=2.5
>2hours and <=3hours	High	<2.5
>2hours and <=3hours	Medium	<2.5
>= 1hour and <= 2hours	Medium	>=2.5
>2hours and <=3hours	High	<2.5

Chandrakanth and Seetharam,
Evaluation for Defective Density in All the Right Places,
Indian journal of engineering, 2014, 11(26), 30-37,
http://www.discovery.org.in/ije.htm