ISSN (Print) : 0974-6846
ISSN (Online) : 0974-5645
Indian Journal of Science and Technology, *Vol 8(S4), 96–100, February 2015* DOI: 10.17485/ijst/2015/v8iS4/63111

Analysing Software Metrics for Accurate Dynamic Defect Prediction Models

Shilpee Chamoli*, Gil Tenne and Sanjay Bhatia

Amdocs, Pune - 411013, Maharashtra, India; SHILPEEC@amdocs.com, Sanjay.Bhatia@amdocs.com, GILT@Amdocs.com

Abstract

Defect Prediction Models are widely used in many software products. These estimates help assessing the risks of defect leakage objectively. However, Most of the published Defect prediction models do not necessarily fit in providing estimates for large complex telecom billing solutions. We have tried to analyze the accuracy of the Prediction model based on various software metrics. It is important to understand the relation between different software metrics for accurate software defect prediction. The Telecom Billing solutions have complex applications, configurations, size and parameters.

We have tried to correctly identify the Software Metrics that affect the accuracy of the software Defect prediction models. This paper presents an evaluation; software metrics are investigated in order to identify the ones which significantly impact the accuracy of defects prediction models. We found that Historical Defect Density, Removal rates can help in accurate Defect prediction.

Data from various projects is collected and analyzed. The prediction of the models, with and without metrics is compared. To access whether the improvements are significant or not, analysis on the accuracy of the Defect prediction is done. It was found that metrics such as Defect Density, Removal rates, trend of absolute number of defects along with Effort for a particular application, significantly affects the accuracy of a Defect prediction tool. Also behavior of one application may or may not be similar to another application. Thus, to have an accurate prediction, we should attempt at avoiding generalization.

Keywords: Defect Prediction Model; Software Metrics for Defect Prediction Model; Software Defect Prediction for Telecom Billing Solutions

1. Introduction

A software defect is a deficiency in a software product that causes it to produce incorrect or unexpected results. In software development projects, a "mistake" or "fault" can be introduced at any stage during development. Mistakes or faults are a consequence of the nature of human factors in the programming task. More complex bugs can arise from unintended interactions between different parts of a computer program. This frequently occurs because computer programs can be complex - millions of lines of code - often having been programmed by many people over a great length of time, so that the programmers are unable to mentally track every possible way in which parts can interact. Defects are Inevitable in the software. They may be there in the software either due to human or non-human factors. Thus, we may say that defects get into the software as a result of the several interacting interfaces – code getting developed under different people (with different skill sets), data getting transferred, Multiple Environment etc.

Defect prediction is important as a means to control the software development project and to ensure that the project meets its targets of quality, cost and time. During the testing phase, sufficient number of developers needs to be assigned to analyze and fix the defects; sufficient number of testers needs to be assigned to re-test the fix. Failing to provide the right resources will cause

*Author for correspondence

either leakage of defects to production, or a delay in the timeline of the project. Moreover, predicting how many defects exist in the code and how many are expected to be detected during testing, will provide the customer with the confidence that the software will function as expected after it is deployed to the users.

Metrics related to quantifying the number of defects are important, as it relates to the quality of the software produced. Thus any effort in predicting the accurate no. of defects in the software will help in predicting the quality of the software and also improving it. If, we are aware of the accurate number of the defects in the software, then we can plan for the resources, time, effort required to solve those defects. Also, if we are unable to find the predicted number of defects, we may have to run an analysis to understand the reasons for same.

If we look at the Brief History of the Defect Prediction studies in about last 50 years, we will find several kinds of Models proposed at various points. The first study estimating the number of defects was conducted by Akiyama[1]. Based on the assumption that complex source code could cause defects, Akiyama built a simple model using lines of code (LOC) since the number of LOC might represent the complexity of software systems. However, only using LOC is too simple to show the complexity of systems. Thus, McCabe and Halstead proposed the cyclomatic complexity metric and Halstead complexity metrics in 1976 and 1977 respectively[2]. These metrics were very popular to build models for estimating defects in 1970s and the early of 1980s[14]. But, these models were not validated on new software modules. To resolve this limitation of previous studies, Shen et al. built a linear regression model and test the model on the new program modules. However, Munson et al. claimed that the state-of-the art regression techniques at that time were not precise and proposed a classification model that classifies modules into two groups, high risk and low risk. The classification model actually achieved 92% of accuracy on their subject system. However, Munson et al.'s study still have several limitations such as no metrics for object-oriented (OO) systems and few resources to extract development process data[4,14].

In terms of Object Oriented systems, Chidamber and Kemerer[10,5] proposed several object-oriented metrics in 1994 and was used by Basili et al[11]. to predict defects in object-oriented system. In 1990s, version control systems were getting popular, development history was accumulated into software repositories so that various process metrics were proposed from the middle of 2000s[5,14].

In 2000s, there had been existed several limitations for defect prediction. The first limitation was that that there were several code changes in the software and it would be more helpful if we can predict defects whenever we change the source code. To make this possible, Mockus et al[7]. proposed a defect prediction model for code changes. Recently, this kind of models is called as just-in-time (JIT) defect prediction models. JIT prediction models have been studied by other researchers in recent years[6].

The second limitation is that it was not possible or difficult to build a prediction model for new projects or projects having less historical data.

The third limitation was from the question, "are the defect prediction models really helpful in industry?" In this direction, several studies such as case study and proposing practical applications have been conducted. There have been several studies following the trends of information technology as well. By using social network analysis and/or network measures, new metrics were proposed by Zimmermann et al[9,14,8].

But, apart from being accurate, these models should also be easy to use. The model should rely only on metrics, which can be easily calculated and used. Some Popular Models Include the models build by Basili et al[11,13,12]; Denaro and Pezze, Gyimothy et al., Tang et al., Design metrics has also been introduced in Chidamber and Kemerer[10]. The goal of this paper is to assess metrics as predictors of defects. Similar work has already been done by Li and Henry, and Lech Madeyski and Marian Jureczko[13,12].

To perform validation accurately, the data has been collected across 8 projects of similar nature of requirements. All eight projects were developed using a waterfall development life cycle model. The validation is done application wise. Based on the quantitative analysis, the advantages and the disadvantages are discussed. Some of the metrics can actually serve as important predictors and their use in Defect Prediction Models can serve in getting accurate predictions.

2. Data Collection

The following attributes impact how many defects are expected. There can be many "Predictors "to find out the probable defect count. It is also important to access,

the sensitivity of each and every "Predictors" and adjust weights accordingly.

2.1 Version Size

Development effort or any other measurement of release size, such as Function Points is in a direct relation to the probable no. of defects in software. The underlying assumption is that the defects are equitably distributed in the sources code and we are not analyzing the code as High risk or low risk. Knowledge pertaining to when and how much code is getting delivered for testing will bring Dynamicity to the prediction (Figure 1).

2.2 Application Type

There are many different Applications in Telecom Billing solutions. The behavior of applications differs from each other as they have different functionalities, and different programming language and complexity.

2.3 Software Development Life Cycle Different Types of Development Life Cycle

Waterfall and Agile have different strategy for the defects getting recorded. The shift from one lifecycle to other kind of life cycle has change in the number of defects getting registered. Agile has closer interaction amongst the Tester and Developers so the generally there are lesser defects registered in agile life cycle as compared to the waterfall.

2.4 Defect Density

The Defect Density of a particular application in a large telecom billing solution may defect from other applications. The historical Defect density of a particular application may serve as an important predictor.

2.5 Defect Removal Rate of Prior Testing Stages

The Removal rates per testing level are different for different application. The dynamic Defect Prediction can benefit from the historical pattern of Removal rates.

The data was collected from several projects to analyze the effect of using metrics.

This section describes the metrics investigated for accurate predictors of the defects. Let us look at some important parameters before analyzing the metrics.

2.5.1 The Historical Data

Firstly, Historical data is of importance as it gives us the precision of the measurement, the probable defects that should be detected during the course of testing. There may be instances when we find lesser or greater no. of defects during the course. The deviation from the normal course would further guide towards reasons for investigation. Also it indicates that we need to conduct a deeper dive before coming on to conclusion, there may be situations such as, lesser number of defects, can be either due to a clean code or poor testing. The Historical data can provide dotted line of reference. Using the historical data from the past projects to create the reference curve, the main points that should be taken care of are - choosing the historical data-similar release (same application), similar effort, and similar length of testing for a fairer comparison.

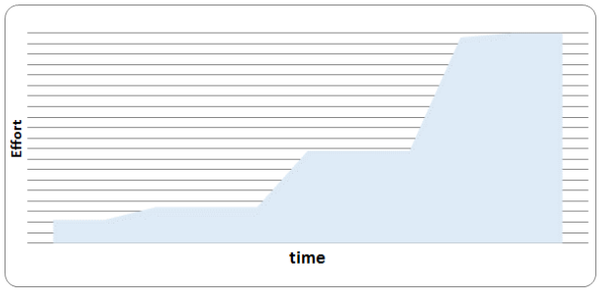

Figure 1. Effort on time.

Secondly, once the dotted line of reference is designed, uses the upper and lower limits to define the probable areas. History also shows us Deviation.

The Metrics that we need to derive using the Historical data and Effort are Density, escaping and removal rates across the stages, in the sequential model. Efforts in Man months along with the Density proportions (historically) will help to find the absolute no. of defects.
The metrics analyzed are:

2.5.1.1 Defect Density

The defect density helps in predicting the overall no of defects. There is a historical relation between the Development effort and Defect density. The data from this trend then helps in predicting the density of future release. Total of 8 projects were considered for this analysis. The correlation value for all the application is not same. For some application, there is a very high correlation between the DCUT and the Density (Figure 2).

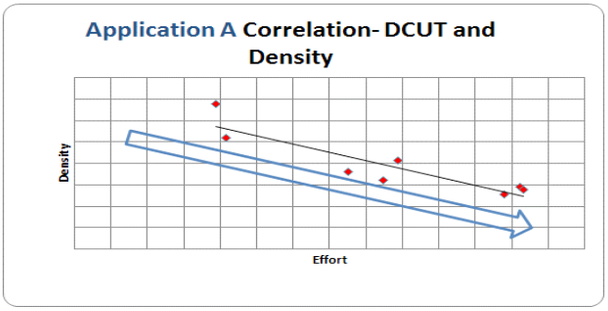

Figure 2. Correlation.

2.5.1.2 Removal Rates

The Removal rates per testing level are different for different application. The dynamic Defect Prediction can benefit from the historical pattern of Removal rates. The actual from the initial stages in the sequential steps can guide in predicting the course of defect detection in the later stages.

Different applications have different trends of the absolute number of defects per Effort. Some phases, such as production, do not have a high impact with the change in Effort, but Testing at the Initial Phases in the sequential cycle has high impact with the change in Effort.

2.5.1.3 Deeper Layer Analysis of the Reasons of Higher Density

When tried to understand the reasons for higher density on lower Development effort, Looking at the analysis of the Root Cause reasons, led us to few reasons for most of the defects in all the projects.

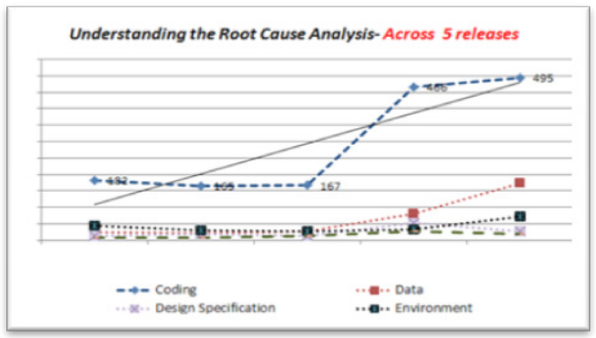

Figure 3. Plot.

In Figure 3, each point is the number of defects found on a version per the phase of defect origin (root cause). Plotting the no. of defects on the graph, with the effort on the X-axis led us to an important finding that only coding related defects are directly related to the increase in the effort. The remaining category of defects (Design, Environment) have no impact and no. of defects due to data issues has low impact with the change in the Development effort.

Thus we can also derive to an important finding that defects density on lower Effort project has a higher impact from the non-coding related defects than on a higher effort project/release:

2.5.1.4 Reference Curve

The historical refernce curve of defect detection varies from application to application, when plotted on the execution time. But, In general it is a "S" Shape curve with leaner detection during the starting and end of the execution phase. It is important to identify the defect detection pattern of the applications.

3. Analysis

The defects data is drawn from defects tracking system (HP-QC), for creating historical reference curves, for all the applications. The defect data of the historical and the current release is map to the actual versus prediction. The data from the past 8 projects is analyzed for 4 applications. The metrics are compared for each application separately.

4. Hypothesis

In order to decide whether the additional metrics are useful in defect prediction, a statistical hypothesis was tested for each of the metrics separately. We built 2 prediction models and compared their results: The Simple Model (M) was built without the advaced set of metrics. Advanced Model (M1) is built using the advaced Metrics (Defect Density Correlation and Removal rates, Application Type Differentaition).

Simple Model - The model is built on the following assumptions - Defect Density improves in 10% from the previous release. Assuming that the software is better quality software; The Developers and the Testers are more knowledgable as compared to the prevoius release. The removal rates is similar to the previous release. Similar Pattern for all Application Type.

Advanced Model - The Model is built on Defect Density Correlation with the Effort. The removal rates are derived from the historical data of the application, as described in chapter II above.

Null Hypothesis

{H0; There is no difference in the accuracy of defect prediction between the simple model (M) and the advanced model (M1).

Alternative Hypothesis

{H1; There is a difference in the efficiency of defect prediction between the simple model (M) and the advanced model (M1).

Both the models are built using a linear equation

Y = M(X)+C

Where in Y(Absolute no. of Defects) = M(Defect Density)X(Effort) + C(additions or removals if any).

The Defects in a particular stage are further broken down as per the Historical removal rate of that stage.

6. Conclusion

The Null hypothesis is rejected. There is a difference in the efficiency between simple model (M) and the advanced model (M1). The advanced model is significantly accurate in predictions (per application basis) as compared to the simple model.

7. References

1. Akiyama F. An example of software system debugging. Proceedings of the International Federation of Information Processing Societies Congress. 1976; 71:353–79.
2. McCabe T. A complexity measure. IEEE Trans Software Eng. 1976 Dec; 2(4):308–20.
3. Halstead MH. Elements of Software Science (Operating and Programming Systems Series). New York, USA: Elsevier Science Inc; 1977.
4. J. C. Munson and T. M. Khoshgoftaar. The detection of fault-prone programs. IEEE Trans Software Eng. 1992 May; 18(5):423–33.
5. Chidamber SR, Kemerer CF. A metrics suite for object oriented design. IEEE Trans Software Eng. 1994 Jun; 20:476–93.
6. Fukushima T, Kamei Y, McIntosh S, Yamashita K, Ubayashi N. An empirical study of just-in-time defect prediction using cross-project models. Proceedings of the 11th Working Conference on Mining Software Repositories (MSR 2014). New York, USA: ACM; 2014. p. 172–81.
7. Kamei Y, Shihab E, Adams B, Hassan AE, Mockus A, Sinha A, Ubayashi N. A large-scale empirical study of just-in-time quality assurance. IEEE Trans Software Eng. 2013 Jun; 39(6):757–73.
8. Kim S, Whitehead EJ, Jr, Zhang Y. Classifying software changes: clean or buggy? IEEE Trans Software Eng. 2008 Mar; 34:181–96.
9. Zimmermann T, Nagappan N. Predicting defects using network analysis on dependency graphs. Proceedings of the 30th International Conference on Software Engineering (ICSE '08). 2008. p. 531–40.
10. Chidamber SR, Kemerer CF. A Metrics Suite for Object-Oriented Design. IEEE Trans Software Eng. 1994 Jun; 20(6):476–93.
11. Basili V, Briand L, Melo W. Measuring the impact of reuse on software quality and productivity. Comm ACM. Oct 1996; 39(10):104–16.
12. Henderson-Sellers B. Object-oriented metrics: measures of complexity. Upper Saddle River, NJ, USA: Prentice-Hall, Inc.; 1996.
13. Madeyski L, Jureczko M. Which process metrics can significantly improve defect prediction models? An empirical study. 2014.
14. Nam J. Survey on software defect prediction.