Bug Prediction for an ATM Monitoring Software
Use of Logistic Regression Analysis for Bug Prediction

Ozkan Sari[1,2] and Oya Kalipsiz[2]
[1]*Department of Computer Engineering, Yildiz Technical University, Esenler, Istanbul, Turkey*
[2]*ATM Software Development Department, Provus Bilisim Hizmetleri, Sariyer, Istanbul, Turkey*

Keywords: Software Metrics and Measurement, Software Engineering, Bug Prediction, Logistic Regression Analysis.

Abstract: Software testing which is carried out for the elimination of the software defects is one of the significant activities to achieve software quality. However, testing each fragment of the software is impossible and defects still occur even after several detailed test activities. Therefore, there is a need for effective methods to detect bugs in software. It is possible to detect faulty portions of the code earlier by examining the characteristics of the code. Serving this purpose, bug prediction activities help to detect the presence of defects as early as possible in an automated fashion. As a part of the ongoing thesis study, an effective model is aimed to be developed in order to predict software entities having bugs. A public bug database and ATM monitoring software source code are used for the creation of the model and to find the performance of the study.

1 INTRODUCTION

It is natural that all software has a certain amount of defects. The important thing is to detect and eliminate these defects. Test activities help to achieve this goal, but even after testing it cannot be said that the software is totally defect free. It is also impossible to check all combinations of inputs, outputs and conditions during the test (Khannur A., 2014). Finding and fixing a defect after delivery is often more expensive up to 100 times. (Boehm and Basili, 2001) In order to find the defects that cannot be detected during test activities and detect these defects as early as possible, an effective method should be devised.

Considering the nature of the software development process, bugs are always possible and hard to detect. The time and developers are also limited and there is not much resource to find bugs and fix them. Thus, bug prediction has been an area of interest for a long time and several academic studies have been performed on this area of research.

Throughout this paper, we use the terms defect and bug synonymously. A fault or defect is a flaw in the software, which causes it to behave incorrectly. Most bug prediction models reports which software entities are likely to contain faults and which are likely to be defect free. Those models are referred as classification models (Weyuker, E. J. et. al., 2010).

The purpose of this study is to find faulty software files/classes by examining the specific metrics of the code and evaluating them with logistic regression analysis in order to form a model.

The developed logistic regression model is wanted to be simple and easy to be used in real software development projects. After the model is formed, it is used to detect bugs in the source code of an Automated Teller Machine (ATM) monitoring system project, which is actively used and still being developed according to the customer needs. This monitoring system is selected instead of larger and more elaborate software, because of the owner company's demands and support for such a case study.

This paper is organized as follows. In section II, we introduce some bug prediction approaches which are related to our study. Section III is about data analysis techniques, particularly about logistic regression. In section IV, the data set used to create the logistic regression model is introduced. In Section V, how logistic regression model is created is explained. Section VI is about the use of proposed model in real software development project data and demonstrates the performance of the developed logistic regression model. Finally, Section VII concludes the paper with some remarks and notes

Sari O. and Kalipsiz O..
Bug Prediction for an ATM Monitoring Software - Use of Logistic Regression Analysis for Bug Prediction.
DOI: 10.5220/0005382803820387
In *Proceedings of the 17th International Conference on Enterprise Information Systems* (ICEIS-2015), pages 382-387
ISBN: 978-989-758-097-0

about possible future works.

2 BUG PREDICTION APPROACHES

Different bug prediction approaches propose different kind of methods to solve this problem. Some of the many prediction techniques related to our study are source code metrics, process metrics and previous defects.

2.1 Source Code Metrics

Metrics like cohesion, coupling, complexity, lines of code and object-oriented metrics are used as an instrument to measure source code properties to promote software quality. Most of the bug prediction approaches utilizes the source code metrics. CK metrics (Chidamber and Kemerer, 1994) and object-oriented metrics are some of the well-known source code metrics. Some of these metrics used in this study, are given in Table 1.

Table 1: CK & OO Metrics Used in this study.

Name	Description
wmc	Weighted Method Count
dit	Depth of Inheritance Tree
lcom	Lack of Cohesion in Methods
noa	Number of attributes
nom	Number of methods

2.2 Process Metrics

This approach states that the changes in the source code leads to bugs (Moser et al., 2008). The metrics such as number of revisions, number of authors who make changes, average lines added/removed in a revision are used in process metrics.

Table 2: Metrics used by Moser et al.

Name	Description
NR	Number of revisions
NFIX	Number of times a file has been a part of fix
NAUTH	Number of authors who has previously changed the file
LINES	Number of lines added or removed

The changes in the source code are mostly analyzed from a software configuration management (SCM) system. For instance, NFIX metric represents the number of bug fixes performed on the file, which is extracted from the commit comments of a versioning system like SVN or CVS.

Some of the process metrics related to our study is given in Table 2.

2.3 Previous Defects

By inspecting previous defects in a source code fragment, future defects might be predicted. Zimmermann proposed that the number of past bug fixes correlates with the number of future fixes (Zimmermann T. et al., 2007).

In our study, we focused on whether or not the class file has a previous bug or not, instead of the exact number of bug fixes in the class file. That's because, we realized that a bug fix might be committed in several different commit steps and in such cases the number might be misleading. As given in Table 2, the "BUGS UNTIL" metric denotes if the file had a past defect and the "BUGS AFTER" metric denotes if the file has a bug after the inspected SVN version.

3 DATA ANALYSIS

Data analysis is one of the many techniques which can be used for defect prediction. As a part of this study, regression analysis and specifically logistic regression analysis is used to create a model to estimate the relationship among source code metrics and bug probability.

3.1 Linear Regression Analysis

Regression analysis is used for predicting values or assessing the effects of one or more response variables from a collection of predictor variable values. This technique shows the casual relationship between events and factors (Johnson, R. A. et. al., 2007). One can observe the casual relationship in several incidents in life. For example, the market value of a house might be explained with living area in square feet, location and so on.

The classical linear regression model with a single response variable Y with predictor variables z_i and a random error ε takes the following form:

$$Y = \beta_0 + \beta_1 z_1 + \ldots + \beta_r z_r + \varepsilon \quad (1)$$

The response variable Y is also called dependent variable, and z_i's are independent variables.

Table 3: A sample from the bug prediction data set.

File / Class name	wmc	dit	lcom	noa	nom	…	BUGS UNTIL	BUGS AFTER
IndexBinaryFolder	20.0	2	15	1	6.0	…	1	0
CachedIndexEntry	1.0	1	0	2	1.0	…	0	0
ASTNode	176.0	1	190	131	20.0	…	1	1
MemberTypeBinding	12.0	6	10	0	5.0	…	1	0
CodeSnippetParser	115.0	2	820	7	41.0	…	1	0
…	…	…	…	…	…	…	…	…

3.2 Logistic Regression Analysis

Different from linear regression, logistic regression analysis is a classification technique which is not based on all quantitative variables but some or all of the variables are qualitative. In its simplest setting, the response variable Y in logistic regression model is restricted to two values, which can be coded as 0 and 1 (H. Okamura et. al., 2010). General form of the logistic regression model is as follows:

$$\log \frac{p(x)}{1-p(x)} = \beta_0 + x.\beta \quad (2)$$

The probability of belonging to a category is found using the following equation:

$$p(x;b,w) = \frac{e^{\beta_0 + x.\beta}}{1 + e^{\beta_0 + x.\beta}} \quad (3)$$

The equations in (2) and (3), is used to form a logistic regression model from the selected data set.

4 DATA SET

The bug prediction dataset by D'Ambros et al. is used in our study. The dataset contains data about some popular java-based software systems, namely, Eclipse JDT Core (http://www.eclipse.org/jdt/core/), Eclipse PDE UI (https://eclipse.org/pde/pde-ui/), Eclipse OSGI Equinox Framework (http://eclipse.org/equinox/framework/), Apache Lucene (http://lucene.apache.org/) and Eclipse Mylyn (http://eclipse.org/mylyn/). The database contains metric values of 5371 java class files (D'Ambros et. al., 2010).

The dataset includes the following pieces of information for each class:

- Values of 17 source code metrics (CK + 11 object oriented metrics)
- Values of 15 metrics computed from CVS change log data
- Categorized (with severity and priority) post-release defect counts

Only the metrics suitable to the ATM Monitoring software and can easily be retrieved from its source code are used in our study but all the samples are used to create a logistic regression model to predict whether a class has a defect or not. A small sample of data is given at Table 3.

5 APPLICATION

The data set explained in section IV is used in logistic regression analysis. 10% of the data are spared for tests, 90% of them are used in the analysis.

Table 4: Logistic Regression Model Calculations.

	Estimate	Std. Error	Z Value	Pr(>|z|)	
(Int.)	-3.4680919	0.2484656	-13.9580000	< 2e-16	***
WMC	0.0062890	0.0011536	5.4510000	5.00e-08	***
DIT	-0.0702946	0.0347372	-2.0240000	0.04301	*
LCOM	-0.0001523	0.0000478	-3.1860000	0.00144	**
NOM	0.0248885	0.0056447	4.4090000	1.04e-05	***
NOA	0.0225958	0.0039736	5.6860000	1.30e-08	***
BUG	1.4098361	0.2428095	5.8060000	6.39e-09	***
Significance: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1					
Error ~ WMC+DIT+LCOM+NOM+NOA+BUG					

Logistic regression model is created to find the probability of the existence of a defect/bug in a class file. Variables and their estimates, along with standard error and significance of the model are given in Table 4. Calculations have been performed on the statistical computing tool R (https://www.rstudio.com/).

The problem statement of this study is that, given a version X of a software system S, to find for each class of X, the presence of defects reported after version X until a future version Y.

Logistic regression model equation obtained as a

result of calculations on the statistical computing tool R-studio is as follows:

$$Y = logit(p) = -3.4680919 + 0.0062890 * WMC - 0.0702946 * DIT - 0.0001523 * LCOM + 0.0248885 * NOM + 0.0225958 * NOA + 1.4098361x * BUG \quad (4)$$

By using the value Y in equation 4, one can calculate the probability of a file/class to be classified as faulty as in equation 5.

$$p = \frac{\exp(Y)}{1 + \exp(Y)} \quad (5)$$

The logistic regression model here is validated by the spared 10% test data and the success rate of the model is calculated as 84.642%, as given in Table 5.

Table 5: Logistic Regression Model Test Results.

Sample	573
True Classification	485
False Classification	88
Success Rate	84.642%

6 USE OF THE MODEL

The next step in our study is to use the logistic model in real software development project data, which is the source code from the ATM Monitoring Software, developed in *Provus Bilisim Hizmetleri* in Istanbul, Turkey. In order to use the model in a private source code base, the metrics compatible with the model must be generated and the success of the model must be validated.

6.1 ATM Monitoring Software

ATM Monitoring System is complex and online software of several different modules and components, which gives service to different banks in Turkey, Albania and Cyprus. Software code has been developed using Java based technologies since 2004 and it consists of more than 250.000 lines of code. In our study, only File Distribution Project component source codes are used.

6.2 Data Collection Process

Data collection process is performed by inspecting source codes kept in an internal SVN (https://subversion.apache.org/), a SCM repository. In order to compute the metric values for different Java classes, Eclipse IDE (http://www.eclipse.org), Google Code Pro Analytix Tool (https://developers.google.com/java-dev-tools/codepro/) and Eclipse Metrics Plugin (http://metrics2.sourceforge.net) has been used.

Figure 1 summarizes how the metric values are gathered. SVN repositories (SCM) keep the current status of project files and also commit transaction information. A commit transaction is a set of files which were modified, deleted, or added to repository. These transactions also include version number, date, time, developer and the comment.

As the nature of Java programs, a source file might consists of one main java class and several java inner classes. Only java main classes are considered as a part of this study.

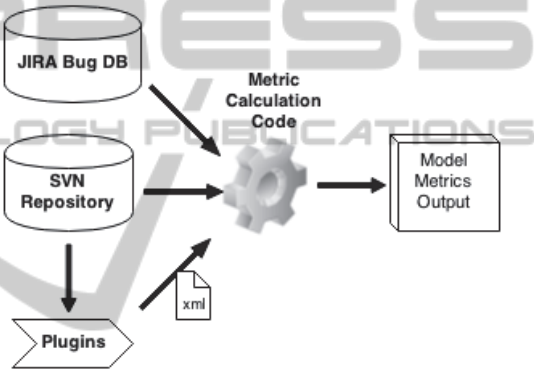

Figure 1: Metric Data Collection.

6.2.1 Calculation of Metric Values

Source code retrieved from SCM is analyzed by *Eclipse Metrics* plug-in and *Google Code Pro Analytix* tool. These tools output metric values for each individual java class as separate xml files. Additional metric values such as previous defects and change metrics are directly gathered from source files in SVN repository. To access the SVN repository and parse the commit transactions and their comments, SVN Kit library (svnkit.com) has been used.

As given in Figure 2, a specific SCM version has been fixed as the prediction version in order to calculate metric values for source code. Current version has also been fixed as validation version to test the performance of the bug prediction calculations.

In order to parse and convert this gathered information as a usable format, a metric calculation program has been developed. The xml files which include the metric values and additional source file

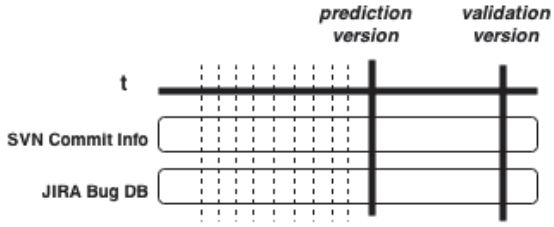

Figure 2: Code Retrieval Approach.

information is input by this program and metric values suitable to be used in logistic regression model equation given in (4) is generated.

6.2.2 Linking Bugs to Classes

As Zimmermann (Zimmermann T. et. al, 2007), Fischer (Fischer M. et. al, 2003) and Bird (Bird et. al., 2009) studied linking bugs to versioning system files, we performed a similar approach and detected bugs by some keywords from SVN commit comments.

In order to find if the java class had a defect which was fixed afterwards, developer comments in SVN commit transactions have been parsed and checked against some patterns to mark the change as a part of bug fix. To be recognized as a bug fix, the commit comment must include some special keywords, such as “fix”, “bug” and some words in Turkish Language specifying a bug fixing activity.

Some additional metrics such as the number of developers that had previously changed the java file is also retrieved by checking SVM commit transactions.

6.3 Performance of the Model

After calculating the metric values required for the logistic regression equation, probability of a class having a defect is predicted. Predicted percentage values more than 50% are accepted that it means to have defect in the inspected class.

In order to validate the predictions, SVN transactions after the prediction version are controlled and it is checked that whether or not any bug fixed or defect detected. This information is specified with the “BUGS” variable.

A portion of the real software data with the required metric values, the prediction and validation result (bugs) are shown in Table 7.

Using the logistic model equations given in Equation 4 & 5, the probability of java classes in ATM Monitoring Software having defect is calculated and this results are validated as explained above. The results, as shown in Table 6, show that the generated logistic regression model is 66.6% successful in the real software data.

Table 6: Performance Results of the model.

Sample	48
True Classification	32
False Classification	16
Success Rate	66.6%

7 CONCLUSIONS

In this study, a logistic regression model is generated by using the bug prediction dataset by D'Ambros et al., which consists of 5371 samples from 5 different popular open-source projects.

To use this model in a selected real software development project, the required metric values are obtained by analyzing the project source code and a small database of metric values is formed. An ATM Monitoring System which belongs to the company *Provus* who supports this study is selected for this study.

Then, the model is applied to the files in project source code and bug probabilities are predicted. Finally, these results are validated by checking the comments in SCM commit transactions after the selected prediction version. The results show a 66% success rate in the selected project source code.

Table 7: Sample Data from Real Software Data.

File/Class	dit	wmc	lcom	noa	nom	BUGS UNTIL	**Prediction**	**BUGS AFTER**
PAYSFDSHelper.java	1	17	0	1	5	1	**13.30%**	**0**
PAYSFDSScheduler.java	1	6	0	2	4	0	**3.37%**	**0**
FDSMonitor.java	0	5	0	0	5	0	**3.52%**	**0**
FDSJobExecutor.java	0	1	0	0	1	0	**3.12%**	**0**
ObjectDefinition.java	1	19	0.571	3	7	0	**4.00%**	**1**
RunnerComparator.java	1	9	0	0	1	0	**3.06%**	**0**

In future, we plan to create some different logistic regression models and compare them with one another and classify their strength and weaknesses. Furthermore, we will examine more samples from real software data and form a larger bug database, which might be published online.

ACKNOWLEDGEMENTS

Authors would like to thank the company of *Provus Bilisim Hizmetleri* in Istanbul, Turkey. The source code of the ATM Monitoring system which is used in this study belongs to *Provus* and *Provus* may not agree with all of the interpretations/conclusions of this paper.

REFERENCES

Weyuker, E. J.; Bell, R. M.; Ostrand, T. J., "We're Finding Most of the Bugs, but What are We Missing?," Software Testing, Verification and Validation (ICST), 2010 Third International Conference on , vol., no., pp.313,322, 6-10 April 2010.

Khannur, A., Structured Software Testing: The Discipline of Discovering, Partridge Publishing, 2014, pages 22-23.

Boehm, B., Basili V. R., "Software Defect Reduction Top 10 List", IEEE Computer, Vol. 34, No. 1, Jan 2001, pages 135-137.

Shyam R. Chidamber and Chris F. Kemerer. A metrics suite for object oriented design. IEEE Trans. Software Eng., 20(6):476-493, 1994.

Raimund Moser, Witold Pedrycz, and Giancarlo Succi. A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction. In Proceedings of ICSE 2008, pages 181-190, 2008.

Thomas Zimmermann, Rahul Premraj, and Andreas Zeller. Predicting defects for eclipse. In Proceedings of PROMISE 2007, page 76. IEEE CS, 2007.

H. Okamura, Y. Etani and T. Dohi, "A multi-factor software reliability model based on logistic regression," Proceedings of 21st IEEE International Symposium on Software Reliability Engineering (ISSRE'10), pp. 31-40, IEEE CPS (2010).

Kuwa, D.; Dohi, T., "Generalized Logit Regression-Based Software Reliability Modeling with Metrics Data," Computer Software and Applications Conference (COMPSAC), 2013 IEEE 37th Annual , vol., no., pp.246,255, 22-26 July 2013.

Johnson, R. A., Wichern D. W., Applied Multivariate Statistical Analysis, Pearson, 6th Edition, April 2, 2007.

Marco D'Ambros, Michele Lanza, and Romain Robbes. An extensive comparison of bug prediction approaches. In MSR '10: Proceedings of the 7th International Working Conference on Mining Software Repositories, s. 31-41, 2010.

Michael Fischer, Martin Pinzger, and Harald Gall. Populating a release history database from version control and bug tracking systems. In Proceedings of ICSM 2003, pages 23-32. IEEE CS, 2003.

Christian Bird, Adrian Bachmann, Eirik Aune, John Duffy, Abraham Bernstein, Vladimir Filkov, and Premkumar Devanbu. Fair and balanced?: bias in bug-fix datasets. In Proceedings of ESEC/FSE 2009, pages 121-130, New York, NY, USA, 2009. ACM.