Evaluation the Quality of Software Design by Call Graph based Metrics

Sanjeev Kumar Punia[1], Dr. Anuj Kumar[2] and Amit Sharma[3]

[1] NIMS University, Jaipur

Received: 6 December 2013 Accepted: 2 January 2014 Published: 15 January 2014

Abstract

The prediction of software defects was introduced to support development and maintenance activities to improve the software quality by finding errors early in the software development. It facilitates maintenance in terms of effort, time and more importantly the cost prediction for software evolution and maintenance activities.In this paper, we evaluate the quality related attributes in developed software products. The software call graph model is also used for several applications in order to represent and reflect the degree of their complexity in terms of understandability, testability and maintainability efforts. The extracted metrics are investigated for the evaluated applications in correlation with bugs collected from customers bug reports. Those software related bugs are compiled into datasets files to use as an input to a data miner for classification, prediction and association analysis.Finally, the analysis results is evaluated in terms of finding the correlation between software products bugs and call graph based metrics. We find that call graph based metrics are appropriate to detect and predict software defects so that the activities of testing and maintenance stages become easier to estimate or assess after the product delivery.

Index terms—

1 Introduction

he human abilities and creativities play a significant role in producing and directing the software products in software development life cycle with the help of the tools and methodologies. However, humans are also the main source of the errors and defects that occur in the software and discovered before or after the delivery of the product. The production of defect free software and projects is impossible. However, software developers struggle to keep such possible defects at minimum. Finding and fixing the defects and errors after delivery usually cost a large amount of the project budget and resources specially if compared with detecting them earlier. As try to predict the defects early is valuable specially if detected before the delivery of the software where that can also help the project to success in terms of cost and quality.

The coupling metrics play an important role in many software development and maintenance activities such as effort estimation, improving the quality of the software products, test planning and reducing future maintenance. These metrics assess the software quality by supporting the quality related factors after evaluating error proneness, changeability and reusability. The most relevant tools are available as independent or the part of a development environment to compute the coupling metrics statically by tracing possible problems in the source code.

The call graphs metrics represent the relationship between the modules and reflect the degree of complexity of the software. It also helps to find some software metrics such as coupling and cohesion metrics. In general, one way to reduce cost through defects prediction is by using software metrics. Particularly the call graph based metrics is used to predict and improve possible problems in software design and in coding finally.

In this research, we tried to evaluate the effectiveness and power of call graph based metrics in prediction and detection the defects in software products. A tool is developed to generate call graph attributes and metrics by using open source projects. We select three applications as J Edit 4.2, Velocity 1.4 and Velocity 1.6 based on two factors (i) open source projects and (ii) these projects include users bug reports for actual evaluation of the software products. This paper include, the programmed and evaluated call graph based metrics as LOC, Fan In, Fan Out, SGBR and IFC. The LOC, Fan In and Fan Out metrics are known and popular while SGBR and IFC not so popular but after that also we implement in our tool.

This paper is organized into six sections as follows: Section 2 introduces topic and research related studies. Section 3 describes the methodology steps. Section 4 presents the adopted analysis and evaluation measurements. Section 5 shows the conducted results of the experiments and finally Section 6 presents the conclusion and inference from the paper. Abstract-The prediction of software defects was introduced to support development and maintenance activities to improve the software quality by finding errors early in the software development. It facilitates maintenance in terms of effort, time and more importantly the cost prediction for software evolution and maintenance activities.

2 II.

3 Literature Review

In this paper, we evaluate the quality related attributes in developed software products. The software call graph model is also used for several applications in order to represent and reflect the degree of their complexity in terms of understandability, testability and maintainability efforts. The extracted metrics are investigated for the evaluated applications in correlation with bugs collected from customers bug reports. Those software related bugs are compiled into datasets files to use as an input to a data miner for classification, prediction and association analysis.

Finally, the analysis results is evaluated in terms of finding the correlation between software products bugs and call graph based metrics. We find that call graph based metrics are appropriate to detect and predict software defects so that the activities of testing and maintenance stages become easier to estimate or assess after the product delivery.

call graph based dependency metrics to improve the software quality by providing information for defect prediction and estimation. We list some related work in each step that has taken in our project and developed tool in the following sections.

4 a) Call Graph Model

Many researchers studied software modeling and found that modeling techniques are grouped into broadly two categories as (i) graphical modeling techniques and (ii) textual modeling techniques. Graphical modeling technique use a diagram with named symbols that represent the components, the symbols connecting arcs represent the relationship and other notations to represent the constrains. Textual modeling technique use standardized notations and keywords to define major aspects of software product call graph. J. Dollner and Bohnet et.al. [1] Considered the extracting of process call dependencies as one of the most important step in the reengineering process. Therefore they built a tool based on OINK framework for call graph extraction. In addition, the tool also provides a set of hierarchal data, call type information methods definitions and output this information to impor Table formatted file. D. Reniers et.al. [2] Made an enhancement in hierarchical edge bundling (HEB) technique and named candidate visualization (CV) technique in their framework. So they build an experiment to compare their enhancement hierarchical edge bundling and tulip graph visualization framework with several large systems like Bison, Mozilla Firefox, OINK and conclude that hierarchical edge bundling scheme perform better in typical comprehension tasks. M. Jahromi and E. Honar et.al. [3] introduced a new framework for call graph construction for program analysis. They choose ASM and soot a byte code reader for their environment to store information about the structure of the codes such as classes, methods, files and statements.

They also proposed a framework where three algorithms have been implemented for call graph construction i.e. CHA, RTA and CTA and finally they conclude by an experimental study on a verity of source code programs by comparing two byte code reader.

5 b) Code Metrics Extraction

By analyzing both the source code of any software and extracting code metrics is considered as the main preprocess for the reengineering operation. This information provides a clear view about the complexities and difficulties of the software as well as divides the milestones tasks into phases in order to start the reengineering process easily. On the other hand many researchers considered the code metrics and the system complexity information as a good defect tracker.

They setup a number of hypothesis related to defect probability and code metrics to prove the correlation between them but the hottest topics in this research is to define the set of metrics that we can considered them as the optimal defect predictor. The researchers shows many studies to define this set of metrics and try to view it's set as the perfect one that give justifications for their results. It is also find that code metrics, which plays a major role in many research fields and many tools deployed to handle extraction using different approaches.

F. Abreu and Baroni et.al. [4] Presented a new framework for metrics extraction by modeling the extraction data using UML Meta model called FLAME. They briefly mentioned the main characteristics of FLAME for fact extraction and recommended to use in firing a new tool for metrics extraction. The authors introduce an approach to formalize the metrics design in the optimal way where FLAME functions are used to extract well known sets of metrics as MOOD, MOOD2, MOOSE, EMOOSE and QMOOD metrics.

6 c) Defect Prediction from Source Code Metrics and System History

A number of approaches have been deployed for defect prediction based on different criteria and information. Some researchers turn to find bugs in software code by analyzing software source code and compute its complexity. They extracted call graph based metrics from source code and used to decide which part or module of the software code likely to be defected. While other researchers prefer to use the system history to decide which part of them has a big defect probability especially when the application has many releases. They find that the system history is more accurate to predict defected parts of the system more than the code complexity extraction predict. While some studies support the two approaches together and use both in finding systems bugs.

A. Bernstein et.al. [5] compare the influence of different metrics used in defect prediction and defect prediction densities by using decision tree learners. They collected the needed data that is source code metrics and bugs report in the experiment from seven versions of open source code for Mozilla application. They applied J48 algorithm in WEKA data miner on the data set and setup a number of experiments to test their purposed hypothesis on defect predicting in software parts. They conclude that a simple tree learner can produce good results with various sets of input data.

N. Nagappan and T. Ball et.al. [6] introduce a new technique for prediction defect density by using code churn measures. The idea was drawn with a hypothesis that if code changes many times in the prerelease version then it also has a big chance to be defected in the post release. The authors build an experiment on W2K3 release with its service back and showed with its result that code churn is a good defect predictor. As they noticed that the increase of the code churn measures leads to an increase on the defect density in any software so they conclude that their metrics suit with line of code churned, deleted line of code, files churned, churn count and weeks of churn.

The aim of software developers is to evaluate the cost and quality of software before deliver to customers so that they can predict and find bugs or defects especially for critical systems.

7 III.

8 Methodology

Our methodology consists of six main phases as shown in Figure 1. This begin by phase 1 begin by "implementation of metric computing tool" to built a tool that can read source code of an application to compute some metrics coupling measurements. Phase 2 is "generation of call graph model" that utilizes the application model. Phase 3 is "calculation of call graph based metrics" used to compute some call graph based metrics for our application model. Phase 4 is "generation of data set" used to prepare data set consisted from metrics values for each class in application. Phase 5 is "refinement of data set with bug report" that assigns each data set record with its number of bugs. Finally, phase 6 is "Analysis and Evaluation of Data Set using WEKA" used for the purpose of evaluating its quality and find the correlation between its bug and call graph based metrics.

As the tool focus on extract the call graph based metrics so the developed tool generated data set does not contain bug attribute for each class. This phase is responsible to make some refinement on the comma separated values comma separate value (CSV) file.

Firstly, the tool automatically fill the bug attribute filed for each class by providing its previous comma separate value file for the same application under investigation and contains the bug report for each class. Then by mapping the name of classes between our comma separate value file and the previous worked comma separate value file. The source of previous comma separate value file gained from promise data repository which contains several data sets in comma separated values or attribute relation file (ARF) format. These files are created and prepared by researchers those worked at the topic of software defects prediction. In our research, we use bug attribute for the files which relate to the applications in our experiments.

IV.

9 Analysis and Evaluation

Before you begin to format your paper, first write and save the content as a separate text file. Keep your text and graphic files separate until after the text has been formatted and styled. Do not use hard tabs and limit use of hard returns to only one return at the end of a paragraph. Do not add any kind of pagination anywhere in the paper.

After refine the generated comma separate value file that represent the data set of our research with bug attribute then it is ready to analyze and evaluate using WEKA 3.7.5 tool as data miner. Here we apply J48, logistic model trees (LMT) and support vector machine (SMO) classifier algorithms. The decision tree algorithms are chosen as we want to look at classifiers that were easy to understand and validate the correlation between call graph based metrics and bugs.

10 a) Evaluation Measures

The evaluation process of our testing tool depends on five matrices in term of call graph based metrics measurement. The five matrices are line of code (LOC), Fan In, Fan Out, call graph based ranking (CGBR) and information flow complexity (IFC) as shown in Table 1. The metrics value for each type (LOC, Fan in, Fan Out, CGBR and IRC) depends on the functions that extracted from the application under investigation by which the higher metric value type achieves a higher complexity value. The values of metrics related to class level are computed by find the summation of all corresponded metrics to function level. For example: if we have 10 included functions at such class and each function has Fan In metrics value equal 1 then the class has Fan In metrics value equal the summation of all Fan In metrics values related to functions of the class which equal to 10. The five metrics we use in this research are related to size of the software or coupling and dependency between the components and functions of the application under investigation. LOC metric value represents the number of execu Table and noncommented lines of code. FanIn metric value for such function represents the number of function calling for a given function. Fan Out metric value for such function represents the number of function being called by a given function. CGBR metric depends on the page ranking algorithm that used by almost all the search engines where the ranking methodology is adopted to functions of the software.

This metric hypothesis that more frequently used functions and less frequently used modules should have different defects and bugs characteristic. In the equation used to compute CGBR value, value of d represents damping factor and refer to probability of such function being called or used and can be computed as the ratio of actual function calls to all possible function calls. CGBR (Ti) is the call graph based rank of module Ti which Call for given function. C(Ti) is the number of outbound calls of function Ti. IFC metric represents the measurement of the total level of information flow for a given function. The value of this type of metric depends on the values of metrics LOC, Fan In and Fan Out for the given function.

11 b) Principle Component Analysis using SPSS

The purpose of this analysis is to show the correlate metrics in developed tool. The PCA analysis for call graph based metrics in developed tool results in 2 orthogonal dimension components were identified from 5 call graph based metrics that have Eigen value more than 1 as shown in Table 2. The variance of Eigen values data set explained by the PC in percent and the cumulative variance are provided for each PC where values above 0.6 are set in boldface. The 2 PCs capture 89.963% of the variance in the data set. The PCs are interpreted as follows:

? PC1: CBGR, LOC, FanIn and FanOut are coupling and size metrics. We have size and coupling metrics in this dimension. This shows that there are classes with high internal methods i.e. methods defined in the class and external methods i.e. methods called by the class. This means coupling is related to number of methods and attributes in the class.

? PC2: IFC measure the total level of information flow of a module and reflect the degree of flow complexity among classes.

12 c) Experiments

At the first step, we collect the source code of the applications for the study i.e. JEdit 4.2, Velocity 1.4 and Velocity 1.6 application. We enter the source code for each application to a developed C# tool in order to generate call graph model for each application. The developed tool computes the call graph based metrics for each extracted function. Then compute the same metrics to classes and output the results into comma separate value file that represent the data set to be tested. The next step is refining the data set with bug report related to each application under investigation.

Finally, evaluate the value of the metrics in terms of bug and defect detection the format of the data set should be ARRF file as the classifier algorithms such as J48 and M5P algorithm accepts only the files with that format. The accuracy is calculated with tenfold cross validation. The attributes of the file listed in the Figure **??**. @attribute ”Number” ”numeric” @attribute ”LOC” ”numeric” @attribute ”Fan In” ”numeric” @attribute ”Fan Out” ”numeric” @attribute ”CGBR” ”numeric” @attribute ”IFC” ”numeric” Figure **??** : Data set attributes The attribute bug is classified into three categories based on the number of bugs for each class as shown in Table 3. The experiments result shows that there is an obvious correlation between the call graphs based metrics, bugs and defects of the application. The result of all the nine experiments is summarizing by Table 4. The correlation between bug and the call graph based metrics will be high when we split the bug class into small number of categories, like category three that split the bug class into two categories. So we take category three as criteria to compare the J48 classifier on the applications under investigation output to other classifier output such as logistic model trees (LMT) and support vector machine (SMO) classifier algorithm.

13 Global

The results of three classifier algorithms have approximately similar values and we conclude that correlation is very high between the call graph metrics and bugs of the application under investigation as shown in Table 5.

Finally, we make some normalization to our data set by excluding the non public functions such as private and protected functions from the computation of the call graph metrics for the applications under investigation and the results of analysis is shown in Table 6. The results of three classifier algorithm are approximately have similar values where that leads us to conclude that correlation is very high between the call graph metrics that computed without non public functions and bugs of the application under investigation as shown in Table 6.

After comparing the results of Table 5 and Table 6, we show that excluding the non-public functions such as private and protected functions in order to compute the call graph based metrics for the classes of the application under investigation will raise the percentage of the supposed correlation between call graphs based metrics and bugs.

V.

14 Conclusion

In this paper, we present the effectiveness and the power of call graph based metrics in prediction and detection the defects in software through our developed tool. We choose three applications J Edit 4.2, Velocity 1.4 and Velocity 1.6. We extract the call graph based metrics such LOC, Fan In, Fan Out, SGBR and IFC from the selected applications and evaluate their correlation according to many categories of bugs for the applications. By all these experiments we discover that how much the extracted call graph metrics are necessary and important in lightening the expensive and time consumer obstacles and problems of software that may arise after delivery phase. Therefore, it will be more effective to predict them and find their solutions earlier before they occur at any time.

The results of our research improve the hypothesis of correlation between call graph based metrics and bugs in software design. The highest percentage of correlation was shown in results of the analysis J Edit 4.2 application using J48 algorithm classifier with metric correlation 86%, while the metric correlation resulted in analysis velocity application with its versions 1.4 and 1.6 was 85% and 72% respectively. In addition, the results show that correlation between bugs and the call graph based metrics will be high when we split the bug class into small number. In addition, the results show that excluding non-public functions such as private and protected functions in order to compute the call graph based metrics for the classes of the application under investigation will raise the percentage of the supposed correlation.

By this approach, we proved that call based metrics are appropriate criteria for helping the maintenance and developing stages to be more effective and less costly at the same time for the systems those are very complex and hardly to understand. [1]

Figure 1: T

1

Metric Type	Measurement of Metrics
LOC	No of execu Table and non-commented lines of code for each function
Fan In	No of calling function list
Fan Out	No of called function list
CGBR	(1-d) + d *?i CGBR(T i) C(T i)
IFC	IFC(M)=LOC(M)+ [Fan In(M)*Fan Out(M)]2

Figure 2: Table 1 :

2

	Component	
	1	2
Eigenvalue	3.475	1.063
% of Variance	69.768	18.672
Cumulative %	68.362	89.235
CGBR	0.923	-0.112
LOC	0.905	-0.131
IFC	-0.036	0.923
FanIn	0.836	0.132
FanOut	0.963	0.021

Figure 3: Table 2 :

3

Bug Categories	Metric Matrix
One	VL = 0 error / L = 1 error / M = 2 error / H = 3 errors / VH => 3 errors
Two	L = 0 error / M = 1-2 errors / H => 2 errors
Three	False = no error / True = error exist

Figure 4: Table 3 :

4

	bug categories		
Bug Category Application Name	Category 1	Category 2	Category 3
JEdit 4.2	81.34 %	80.84 %	86.93 %
Velocity 1.4	60.67 %	72.07 %	80.04 %
Velocity 1.6	66.59 %	67.36 %	73.83 %

Figure 5: Table 4 :

5

algorithm types

Classifier Algorithm Application Name	J84	LMT	SMO
J Edit 4.2	86.142 %	84.926 %	82.547 %
Velocity 1.4	80.928 %	80.364 %	75.723 %
Velocity 1.6	72.152 %	71.029 %	66.487 %

Figure 6: Table 5 :

6

excluding non-public functions

Classifier Algorithm Application Name	J84	LMT	SMO
JEdit 4.2	86.924 %	85.196 %	83.537 %
Velocity 1.4	85.918 %	88.364 %	75.783 %
Velocity 1.6	72.125 %	70.709 %	67.467 %

Figure 7: Table 6 :

[1]© 2014 Global Journals Inc. (US) Evaluation the Quality of Software Design by Call Graph based Metrics

[Abreu and Baroni ()] 'A formal library for aiding metrics extraction'. F B Abreu , A L Baroni . *8th International Workshop on Object Oriented Reengineering* 2013.

[Jahromi and Honar ()] 'A framework for call graph construction'. M Jahromi , E Honar . *Student thesis At School of Computer Science* 2012. (Physics and Mathematics)

[Darbyshire and Prins ()] 'Call graph based program analysis with .Net'. P Darbyshire , W Prins . *Procs of the IRMA International Conference*, (s of the IRMA International Conference) 2012. p. .

[Usmani and Azeem ()] 'Defect prediction leads to high quality product'. S Usmani , N Azeem . *Journal of Software Engineering and Applications* 2011. 4 p. .

[Reniers et al. ()] 'Extraction and visualization of call dependencies for large C/C++ code bases: A comparative study'. D Reniers , A Telea , O Ersoy , H Hoogendorp . *7th IEEE International Workshop on Visualizing Software for Understanding and Analysis*, 2011. 2011. p. .

[Bernstein et al. ()] 'Predicting defect densities in source code files with decision tree learners'. A Bernstein , M Pinzger , P Knab . *Proceedings of the 2011 international workshop on Mining software repositories*, (the 2011 international workshop on Mining software repositoriesShanghai, China) 2011. p. .

[Khare et al. ()] 'Static analysis and run-time coupling metrics'. A Khare , P Batra , M Kaur . *International Journal of Information Technology and Knowledge Management* 2013. 6 p. .

[Ball and Nagappan ()] 'Use of relative code churn measures to predict system defect density'. T Ball , N Nagappan . *Proceedings of the 37th international conference on Software engineering*, (the 37th international conference on Software engineeringSt. Louis, MO, USA) 2012. p. .

[Dollner and Bohnet ()] 'Visual exploration of function call graphs for feature location in complex software systems'. J Dollner , J Bohnet . *Proceedings of 2010 ACM symposium on Software visualization*, (2010 ACM symposium on Software visualization) 2010. 1 p. .