IPASJ International Journal of Computer Science (IIJCS)
Web Site: http://www.ipasj.org/IIJCS/IIJCS.htm
A Publisher for Research Motivation ..... Email: editoriijcs@ipasj.org
Volume 2, Issue 11, November 2014 ISSN 2321-5992

EVALUATING THE VARIOUS CODE METRICS OF OPEN SOURCE SOFTWARE

KOMAL MAHAJAN

GNDU REGIONAL CAMPUS, GURDASPUR, INDIA

ABSTRACT

Software engineering metric is a superb program for characterizing, examining, predicting, and providing different functions for software construction procedures and products. Using the software metric, understanding the similarity in them and the design method can be achieved if measurements are taken on similarity in the software processes and the various software products. In this paper, a discussion has been done on open source software and various metrics like CBO, DIT, LCOM, NOC, RFC ETC. From the survey, it has been found that none of the metric has been very efficient. The paper has been concluded with a future scope to overcome these issues.

Keywords:- Open Source Software, Metrics

1. INTRODUCTION

Open source software is software that could be easily applied, transformed, and shared (in revised or unmodified form) by anyone. Open source software is made by many individuals, and distributed under licenses that adhere to the Open Source Definition. The Open Source Initiative (OSI) is a global non-profit that helps and stimulates the open source movement. Among other activities, it keeps the Open Source Description, and a list of licenses that comply with this definition.

Open source identifies computer software for which:

- The source code can be acquired to the end-user.
- The source code could be revised by the end-user.
- Can find number restrictions on redistribution or use.
- The licensing conditions are meant to aid extended re-use and large option of the software, in both industrial and non-commercial contexts.

There are several different functions which several, but not totally all, open source software products have in common:

- The expense of immediate purchase to the end-user is usually minimal; this is because the best to easily redistribute the software makes offering licenses for copies of open source software an impossible company idea.
- The development system of start source tasks gives several features with Agile development, because produces are frequent, functions are included rapidly following comments from customers, developers tend to be distributed geographically.
- Many, but in no way all, open source tasks are made and maintained by informal areas of developers, consumers and evangelists, as opposed to industrial businesses.
- Open source tasks frequently offer as apprentice opportunities for junior developers to fast understand their business by doing real-world development.

2. VARIOUS SOFTWARE METRICS

Software metrics play a crucial role in the administration of the software projects. Metrics used to track development process, quantify restructuring impact and to calculate code quality. Software metrics are very important to software engineering for measuring software complexity and quality, functionality, Characteristic of the software product. Software metrics can be used for Finding defects in the code, predicting defective code, predicting project success, and predicting project risk. Metrics help to identify, prioritize, track and communicate project issues at all levels. Metrics can accurately describe the status of software project processes and product. Metrics can be used as a resource to anticipate problems and to avoid being forced into a reactive fix. Metrics provide an effective rationale for selecting the best alternatives.

2.1 CBO - Coupling between Objects

Coupling between objects (CBO) is just a count of how many classes which are combined to a particular type i.e. where the methods of 1 type contact the methods or access the parameters of the other class. These calls must be counted in equally instructions so the CBO of type A is how big the pair of classes that type A referrals and these classes that research type A. Because this can be a set - each type is counted only one time even if the research runs in equally instructions i.e. if A referrals W and W referrals A, W is just counted once. C&K viewed that CBO should really be as low as feasible for three causes -

- Higher combining increases interclass dependencies, making the rule less modular and less ideal for reuse. Quite simply if you wanted to deal up the rule for recycle you might end up having to include rule that has been certainly not basic to the core functionality.
- More combining ensures that the rule becomes more difficult to steadfastly keep up since an alteration to rule in one single region runs an increased risk of affecting rule in another (linked) area.
- The more hyperlinks between classes the more complicated the rule and the more difficult it is to check.
- The only factor in calculating this factor is whether just the classes published for the challenge should be considered or whether selection classes should really be included.

2.2DIT - Depth of Inheritance Tree

Depth of Inheritance Tree (DIT) is really a count of the properties that the particular class inherits from root. C&K suggested the following effects based on the range of inheritance –

- The deeper a class is in the hierarchy, the greater how many practices it probably will inherit, rendering it more complex to anticipate its nature.
- Greater trees constitute greater design difficulty, because more practices and classes are included.
- The deeper a particular class is in the hierarchy, the greater the possible sell of inherited practices.

2.3 LCOM - Lack of Cohesion of Methods

LCOM is the absolute most controversial and fought around of the C&K metrics. In their original incarnation C&K explained LCOM on the basis of the amounts of pairs of methods that provided references to instance variables. If the couple don't share references to any instance variable then the count is increased by 1 and when they do share any instance factors then the count is reduced by 1. LCOM is viewed as a way of measuring how properly the techniques of the type co-operate to reach the seeks of the class. A low LCOM price implies the type is more cohesive and is viewed as better. C&K's rationale for the LCOM technique was the following –

- Cohesiveness of methods inside a type is desirable, because it promotes encapsulation.
- Lack of cohesion indicates classes must probably be split into several subclasses.
- Any way of measuring disparateness of methods assists identify weaknesses in the design of classes.
- Minimal cohesion raises difficulty, thereby increasing the likelihood of mistakes during the growth process.

2.4 NOC - Number of Children

Number of Children (NOC) is explained by C&K how many immediate subclasses of a class exists. C&K's see was that

- The higher number of children, the greater the level of reuse, since inheritance is a form of reuse.
- The higher number of children, the greater the likelihood of incorrect abstraction of the parent class. If a class features a big amount of children, it might be an event of misuse of sub classing.
- The amount of children offers a concept of the potential influence a class is wearing the design. If a class features a big amount of children, it may require more testing of the techniques in that class.

2.5 RFC - Response For Class

This is the size of the Response set of a class. The Result collection for a class is described by C&K as some practices that will probably be executed in reaction to an email obtained by an item of the class. Since it is really a collection each named method is counted when regardless of how often it's called. C&K's see was that -

- In case a large quantity of practices could be invoked in reaction to an email, the screening and debugging of the class becomes more complicated since it requires a greater degree of knowledge required on the part of the tester.
- The bigger the number of practices which can be invoked from a class, the higher the difficulty of the school
- A worst event price for possible reactions may support in ideal allocation of screening time.

2.6 WMC - Weighted Methods for Class

Weighted methods for Class (WMC) was actually proposed by C&K whilst the sum of all difficulties of the methods in the class. As opposed to use Cyclomatic Complexity they assigned each method a complexity of one making WMC

equal to the number of strategies in the class. Many traditional implementations follow this rule. C&Ks see of WMC was -

- How many strategies and the complexity of strategies included is really a predictor of simply how much time and energy is required to build and keep the class.
- The bigger the number of strategies in a class the greater the possible affect on children, because children can inherit all the methods explained in the class.
- Classes with many strategies are likely to be more application unique, decreasing the likelihood of reuse.

3. LITERATURE SURVEY

E. Constantinou et al. [1] investigated how design metrics could reveal architectural information about a pc software system and more specifically, how architectural layers were correlated to design metrics. In the last years, software development practices included open source code reuse. Since documentation gave little or no information about the device architecture, a prohibitive quantity of effort was required to comprehend the source code and the entire system architecture. Finally, they presented an empirical study on two large open source systems written in Java, attempted to spot metrics revealing information about the device architecture. S. Eski et al. [2] presented an empirical study about the relation between object oriented metrics and changes in software. To be able to obtain the outcome, they analyzed modifications in software throughout the historical sequence of open source projects. Software quality is an important key factor to determine critical parts since high quality parts of software are less error prone and simple to maintain. As object oriented software metrics give important evidence about design quality, they can help software engineers to select critical parts, which will be tested firstly and intensely. Empirical link between the analysis indicated that the lower level quality parts of a pc software change frequently during the development and management process. By using this relation they proposed a method that may be used to estimate change-prone classes and to determine parts which will be tested first and more deeply. Y. Takai et al. [3] focused on latent faults detected by static analysis techniques. The coding checker was popular to find coding standards violations which are strongly relating to latent faults. Many software metrics predicated on various areas of a product and/or a process have now been proposed. There was some research which discussed the relation between software metrics and faults to make use of these metrics whilst the indicator of quality. Most of these software metrics were predicated on structural options that come with products or process information related to explicit fault. In this paper, they proposed new software metrics predicated on coding standards violations to recapture latent faults in a development. They analyzed two open source projects by using proposed metrics and discussed the effectiveness. R.Premraj et al. [4] presented a replication of one particular study conducted by Zimmermann and Nagappan on Windows Server 2003 where in actuality the authors leveraged dependency relationships between software entities captured using social network metrics to predict whether they were likely to possess defects. They found that network metrics performed significantly better than source code metrics at predicting defects. To be able to corroborate the generality of the findings, they replicated their study on three open source Java projects, viz., JRuby, ArgoUML, and Eclipse. Their effects were in agreement with the original study by Zimmermann and Nagappan when using a similar experimental setup as them (random sampling). However, if they evaluated the metrics using setups more fitted to industrial use -- forward-release and cross-project prediction, they found network metrics to provide no vantage over code metrics. Moreover, code metrics might be preferable to network metrics considering the data was easier to collect and they used only 8 code metrics compared to approximately 58 network metrics. K.Kaur et al. [5] reviewed the design level class cohesion metrics with a unique focus on metrics which use similarity of parameter forms of types of a class as the cornerstone of its cohesiveness. Keeping in mind the anomalies in the definitions of the present metrics, a plan of the NHD metric has been introduced. It was named NHD Modified (NHDM). The metric was analyzed with the mathematical properties of cohesion metrics as proposed in research literature. An automated metric collection tool was used to collect the metric data from an open source software program. Statistical analysis of the data indicated that NHDM metric tok the lowest average value in this group of four metrics. It could be because of the undeniable fact that NHDM, unlike other metrics, did not give any false positives/negatives. S.Voulgaropoulou et al. [6] aimed to execute measurements on the R statistical open source software, examined the relationships on the list of observed metrics and special attributes of the R software and search for certain characteristics that defined its behavior and structure. For this function, a random sample of 508 R packages have been downloaded from the CRAN repository of R and has been measured, using the SourceMonitor metrics tool. The resulted measurements, and also a significant amount of specific attributes of the R packages, were examined and analyzed, leading to interesting conclusions like the validity of an electrical law distribution regarding many the sample's metrics and the lack of specific patterns because of the interdependencies among packages. Finally, the consequences of the number of developers and the number of dependencies were investigated, in order to understand their affect the metrics of the sample packages. Q. Zoubi et al. [7] evaluated the ability of Software source code analysis process and tools to predict possible defects, errors or problem in the software products. More specifically, they

evaluated the aftereffect of improving the code based on recommendations from source code tools on software metrics. Several open source software projects were selected for the case study. The output of applying source code analysis tools on those projects resulted in many forms of warning. After performing manual correction of the warning, they compared the metrics of the evaluated projects before and after applying the corrections. Results showed that the size and structural complexity generally are increases. On the other hand, some of the complexities related to coupling and maintainability are decreases. Y.Saski et al. [8] proposed performing preprocessing for metrics measurement and a methodology of the preprocessing. The proposed preprocessing simplifies repeated structures in source code. Through the use of the proposed preprocessing to metrics measurement, they found low-understandability modules more efficiently. Also, they compared results of metrics measurement with and with no proposed preprocessing on open source software systems. As a result, they confirmed that metrics measurement with the proposed preprocessing was more beneficial to find low-understandability modules than without it. P. Singh et al. [9] presented an empirical study of the fault prediction capabilities of object-oriented metrics written by Chidamber and Kemerer in this paper. Open source software systems were playing important roles in many scientific and business software applications. To make certain acceptable quantities of software quality Open source software (OSS) development process uses advanced and effective techniques. Quality improvement involved the detection of potential relationship between defect and open source software metrics. They've carried out an empirical study and tried to find whether these metrics were significantly connected with faults or not. With this they'd extracted source code processed it for metrics and associated it with the bugs. Finally the fault prediction capabilities of object oriented metrics have been evaluated by utilizing Nai□ve Bayes and J48 machine learning algorithms. A.D. Bakar et al. [10] discussed the character of an Open Source Software development paradigm forced individual practitioners and organization to adopt software through trial and error approach. This result in the issues of finding software and then abandoning it after realizing its not enough important qualities to accommodate their requirements or facing negative challenges in maintaining the software. These contributed by not enough recognizing guidelines to lead the practitioners in selecting out from the dozens available metrics, the most effective metric(s) to measure quality OSS. In this study, the novel results provided the guidelines that result in the development of metrics model that could select the most effective metric(s) to predict maintainability of Open Source Software. S. GalaPerez et al. [11] studied exactly what do be inferred from such a metric for projects of the ASF. They've discovered that the metric appeared to be an intensive metric as it was independent of how big is the project, its activity, or the number of developers, and remain relatively independent of the technology or functional part of the project. Their analysis provided evidence that the metric was related to the technical effervescence and popularity of project, and as such could be a good candidate to measure its healthy evolution. Other, similar metrics -like the ratio of developer messages to commits and the ratio of issue tracker messages to commits- were studied for many projects as well, in order to see if they'd similar characteristics. E. Bouwers et al. [12] executed this analysis for just two architecture level metrics, Component Balance and Dependency Profiles, by analyzing the challenges involved with applying these metrics in a commercial setting. Unfortunately, a structural analysis of the usefulness of metrics in a real-world evaluation setting was often missing. This kind of evaluation was important to comprehend the situations where a full could be applied, to spot areas of possible improvements, to explore general problems detected by the metrics and to define generally applicable solution strategies. Additionally, they explored the usefulness of the metrics by conducting semi-structured interviews with experienced assessors. They documented the lessons learned both for the applying of these specific metrics, along with for the method of evaluating metrics in practice. J. Michura et al. [13] Software developers require data to understand the faculties of systems, such as for instance as an example difficulty and maintainability. In order to more realize and establish faculties of object-oriented (OO) systems, that function has explained study that recognizes qualities which are useful in deciding the problem in utilizing improvements during preservation, as well as the possible consequences that such improvements might produce. Some metrics are proposed to measure and evaluate these attributes. The proposed difficulty metrics has placed on ascertain the problem in utilizing improvements through the measurement of strategy difficulty, strategy range, and difficulty density. The report established affect metrics to ascertain the possible consequences of fabricating improvements to a class and dependence metrics that are accustomed to assess the possible consequences on certain type caused by improvements in numerous classes. B.M. Goel et al .[14] In object-oriented systems, assessing reusability represents a built-in role in lowering a price and increasing the quality of the software. Object oriented programming helps in achieving the idea of reusability through several types of inheritance applications, which more assist in creating reusable software modules. And object-oriented metrics recognize the potency of each delete strategy. Software reusability has significant influence on software quality. But software quality improvement can not be recognized unless it's measured. This report centers on an empirical evaluation of object-oriented metrics in C++ applying three various object-oriented features. This has been unearthed that multilevel inheritance has more affect reusability among these three features. M. Bansal et al. [15] Software metrics play a significant role in creating top quality software as properly as to enhance the developer's productivity. To produce good

quality object oriented applications, emphasis through early stages of software development is necessary. An evaluation of object oriented metrics has been shown for identification and validation of object oriented metrics and out of numerous metrics, a restricted pair of metrics is identified which have an important role in software complexity and quality measurement. R.S. Kenett et al. [16] conducted research in developing methodologies for managing risks of FLOSS adoption and deployment in various application domains. This paper was about the capacity to systematically capture, filter, analyze, reason about, and build theories upon, the behavior of an open source community in conjunction with the structured elicitation of expert opinions on potential organizational business risk. The novel methodology presented blends together qualitative and quantitative information included in a larger analytics platform. The approach combined big data analytics with automatic scripting of scenarios that permits experts to assess risk indicators and business risks in focused tactical and strategic workshops. These workshops generated data that was used to make Bayesian networks that map data from community risk drivers into statistical distributions that have been feeding the platform risk management dashboard. A special feature of the model is that the dynamics of an open source community were tracked using social network metrics that capture the structure of unstructured chat data. The technique was illustrated with a working example based on experience gained in implementing their approach in a academic smart environment setting including Mood bile, a Mobile Learning for Moodle. W. Yangsong et al. [17] quantified the quality of a developer via the percentage of history bug-introduce commits total his/her commits throughout the development process. Then, they leverage developer quality information to develop eight file quality metrics. Finally, they empirically study the usefulness of these eight file quality metrics for fault-proneness prediction. Centered on eight open source software systems, their experiment results show that these proposed file quality metrics captured more information weighed against existing process metrics, nearly all the proposed file quality metrics had an important association with fault-proneness in a expected direction, and the proposed file quality metrics could in general improved the potency of fault-proneness prediction models when together used in combination with existing process metrics. These results suggested that developer quality includes a strong influence on software quality and should be studied into account when predicting software fault-proneness.

4. CONCLUSION AND FUTURE SCOPE

In this paper, a discussion has been done on open source software and various metrics like CBO, DIT, LCOM, NOC, RFC ETC. Open source software is software that could be easily applied, transformed, and shared (in revised or unmodified form) by anyone. Open source software is made by many individuals, and distributed under licenses that adhere to the Open Source Definition. From the survey, it has been found that none of the metric has been very efficient. This work has not proposed any new metric, so in near future; we will propose a new metric which will be based on fuzzy if-then rules. So it will definitely have better results.

References

[1] Constantinou, Eleni, George Kakarontzas, and Ioannis Stamelos. "Towards open source software system architecture recovery using design metrics." In Informatics (PCI), 2011 15th Panhellenic Conference on, pp. 166-170. IEEE, 2011.

[2] Eski, Sinan, and Feza Buzluca. "An empirical study on object-oriented metrics and software evolution in order to reduce testing costs by predicting change-prone classes." In Software Testing, Verification and Validation Workshops (ICSTW), 2011 IEEE Fourth International Conference on, pp. 566-571. IEEE, 2011.

[3] Takai, Yasunari, Takashi Kobayashi, and Kiyoshi Agusa. "Software Metrics based on Coding Standards Violations." In Software Measurement, 2011 Joint Conference of the 21st Int'l Workshop on and 6th Int'l Conference on Software Process and Product Measurement (IWSM-MENSURA), pp. 273-278. IEEE, 2011.

[4] Premraj, Rahul, and Kim Herzig. "Network versus code metrics to predict defects: A replication study." In Empirical Software Engineering and Measurement (ESEM), 2011 International Symposium on, pp. 215-224. IEEE, 2011.

[5] Kaur, Kuljit, and Hardeep Singh. "Towards a Valid Metric for Class Cohesion at Design Level." In Emerging Applications of Information Technology (EAIT), 2011 Second International Conference on, pp. 351-354. IEEE, 2011.

[6] Voulgaropoulou, Sophia, Georgios Spanos, and Lefteris Angelis. "Analyzing measurements of the R statistical open source software." In Software Engineering Workshop (SEW), 2012 35th Annual IEEE, pp. 1-10. IEEE, 2012.

[7] Zoubi, Qosai, I. Alsmadi, and B. Abul-Huda. "Study the impact of improving source code on software metrics." In Computer, Information and Telecommunication Systems (CITS), 2012 International Conference on, pp. 1-5. IEEE, 2012.

[8] Sasaki, Yui, Tomoya Ishihara, Keisuke Hotta, Hideaki Hata, Yoshiki Higo, Hiroshi Igaki, and Shinji Kusumoto. "Preprocessing of metrics measurement based on simplifying program structures." In Software Engineering Conference (APSEC), 2012 19th Asia-Pacific, vol. 2, pp. 120-127. IEEE, 2012.

[9] Singh, Pradeep, and Shrish Verma. "Empirical investigation of fault prediction capability of object oriented metrics of open source software." In Computer Science and Software Engineering (JCSSE), 2012 International Joint Conference on, pp. 323-327. IEEE, 2012.

[10] Bakar, A. D., A. B. Sultan, H. Zulzalil, and J. Din. "Applying evolution programming Search Based Software Engineering (SBSE) in selecting the best open source software maintainability metrics." In Computer Applications and Industrial Electronics (ISCAIE), 2012 IEEE Symposium on, pp. 70-73. IEEE, 2012.

[11] Gala-Pérez, Santiago, Gregorio Robles, Jesús M. González-Barahona, and Israel Herraiz. "Intensive metrics for the study of the evolution of open source projects: case studies from apache software foundation projects." In Proceedings of the 10th Working Conference on Mining Software Repositories, pp. 159-168. IEEE Press, 2013.

[12] Bouwers, Eric, Arie van Deursen, and Joost Visser. "Evaluating usefulness of software metrics: an industrial experience report." In Proceedings of the 2013 International Conference on Software Engineering, pp. 921-930. IEEE Press, 2013.

[13] Michura, John, Miriam AM Capretz, and Shuying Wang. "Extension of Object-Oriented Metrics Suite for Software Maintenance." ISRN Software Engineering 2013 (2013).

[14] Goel, Brij Mohan, and Pradeep Kumar Bhatia. "Analysis of reusability of object-oriented systems using object-oriented metrics." ACM SIGSOFT Software Engineering Notes 38, no. 4 (2013): 1-5.

[15] Bansal, Mukesh, and C. P. Agrawal. "Critical Analysis of Object Oriented Metrics in Software Development." In Advanced Computing & Communication Technologies (ACCT), 2014 Fourth International Conference on, pp. 197-201. IEEE, 2014.

[16] Kenett, Ron S., Xavier Franch, Angelo Susi, and Nikolas Galanis. "Adoption of Free Libre Open Source Software (FLOSS): A Risk Management Perspective." In Computer Software and Applications Conference (COMPSAC), 2014 IEEE 38th Annual, pp. 171-180. IEEE, 2014.

[17] Wu, Yangsong, Yibiao Yang, Yangyang Zhao, Hongmin Lu, Yuming Zhou, and Baowen Xu. "The Influence of Developer Quality on Software Fault-Proneness Prediction." In Software Security and Reliability, 2014 Eighth International Conference on, pp. 11-19. IEEE, 2014