EVALUATION OF AN ALGORITHM OF SOFTWARE DEFECTS OF UNDERSTANDABILITY USING A NEW METRICS OF SOFTWARE

[1]Samundra Singh, [2]Dr. Shivani

[1]Research Scholar, [2] Assistant Professor
[1]Department of CSE
[1,2]Bhagwant University, Ajmer, India

Abstract

Prudence is one of the important features of software quality, as it can affect the stability of software. The cost and reuse of software is also likely to make sense. To maintain software, programmers have to understand the source code. The understanding of source code depends on the psychological complexity of the software, and cognitive abilities are required to understand the source code. The understanding of source code is influenced by so many factors, here we have taken various factors from a unified view. In it we have chosen a rough set approach to calculate comprehensibility based on external identity. Outliers generally have an unusual behavior, here we have taken that the project can be easily understood or difficult to understand. Here we have taken a few factors that underlie understanding, bringing forward an integrated approach to determining understanding. We extracted 20 test case metrics, six developer-related metrics, and two understandable proxies from a white-box test case classification experiment. Based on these matrices, we employed classification and regression algorithms to construct the model to understand the test case. From the experiment, we can conclude that the combined matrices always exhibit better discriminated performance in the classification model as well as a higher correlation in the regression model when compared to a model that only includes test case metrics or developer metrics.

Keywords*:* *Matrix, Software, Classification, Model, Developer etc*.

INTRODUCTION

Software products are expensive. Therefore, software project managers are always concerned about the high cost of software development, and desperately seek ways to cut development costs. One possible way to reduce development costs is to reuse parts from previously developed software. In addition to lower development costs and time, reuse also leads to higher quality of developed products as reusable components ensure higher quality.
When programmers try to reuse code that is written by other programmers, faults can occur due to misunderstanding of the source code. The difficulty of understanding limits reuse techniques. The software development life cycle (SDLC) maintenance phase has a comparatively longer duration than all previous phases, obviously resulting in much greater effort. It has been reported that the amount of effort spent on the maintenance phase is 65% to 75% of the total software development [5].
A software metric is a measure of some attribute of a software or its specifications. Since quantitative measurement is necessary in all sciences, there is a continuous effort by computer science practitioners and theorists to bring about the same approach to software development.

Source Code Understability Metrics

Understanding of software also requires some metrics. Here are some metrics of code under-standability [3] that are used by many organizations. Source code readability, quality of documentation, software should be taken into account when measuring key-practicability.

LOC: A common basis for inference on software projects is LOC (Lines of Code). LOC is used to make time and cost estimates.

Comment Percent: RSM (Resource Permanent Matrix) counts each comment line. The degree of commenting within the source code measures the care taken by the pro- grammer to make the source code and algorithms understandable. Poorly commented code makes the maintenance phase of the software life cycle very expensive.

$$\text{Comment Percent} = \frac{CommentLineCount}{LogicalLineCount} \times 100$$

Logical line count = LOC + comment lines + blank lines
In addition to LOC (lines of codes), we can consider using eLOC (effective lines of code), lLOC (logical lines of code), blank lines of code, and white space percentage metrics.

LEN * Length of Names: If the names of processes, variables, constants, etc. are longer, they are probably more descriptive.

Example: a 'is not a good variable name, is not better, eage is more descriptive in employment.
In addition to the length of the names, sometimes we can consider the average length. Names of variables, functions, constants, etc. are considered. We can also consider the name specificity ratio, because when 2 program entities have the same name, it is possible that they are mixed. UNIQ measures the uniqueness of all names.
UNIQ = Number of unique names / Number of total names

Function Metrics: In this we can measure the number of functions and the lines of code per function. Functions that have a large number of lines of code per function are difficult to understand and maintain. They are a good indicator that the function can be broken into a sub function thereby supporting the design concept that the function must perform a singular discrete action.

Function Count Metric: The total number of functions within your source code determines the degree of modularity of the system. This metric is used to determine the average number of LOCs per function, the maximum number of LOCs per function, and the minimum LOCs per function. Apart from function count, we can also use average lines of code, maximum LOC per function, minimum LOC per function metrics.

Macro Metrics [2]: Macros will make it less difficult to understand and maintain. As macros are expanded before the compilation phase, most debuggers will only see macro names and have no reference as to the contents of the macro, so if the macro is the source of a bug in the system, the debugger will never catch it. This situation can waste many hours of labor.
The number of macros used in a system indicates the design style used to build the system. Systems loaded with macros are subject to the portability problem. The macro LOC metric gives insight into how large macros are in the system. The larger the macro, the more complex its structure and the greater the potential for misbehavior hidden by the macro.

Class metrics: In this we can measure the number of classes and the lines of code can be taken per class. The number of classes in a system indicates the degree of object orientation of the system. Additionally, we set the average lines of code per square, the maximum LOC per square, and the minimum LOC per square.

Code and Data Spatial Complexity [7]: Spatial ability is a term used for processing visual information related to an individual's orientation related to cognitive ability, the location of objects in space, and location.

MOTIVATION

Software engineering is much more specialized with other established branches of engineering because it lacks measuring units, well-accepted measures or metrics for soft-ware development. With the lack of such measurement units, software development and maintenance would have stabilized in craft type models. To overcome this deficiency, study, adoption and further improvement require great experience, skills. Soft-ware can be quantitatively described with the help of metrics and evaluation of the use of tools on projects, productivity and quality. Software therefore helps to control, manage and maintain complexity measures. If you cannot measure it, you cannot control it.

RESEARCH OBJECTIVE

1. We want to evaluate the software understanding of different projects, using different metrics.
2. For evaluation of software understanding, we have followed various approaches.
3. To Evaluation of an algorithm of Software defects of Understandability Using a new Metrics of Software
4. To propose a new metric, which can be used in the evaluation of software understanding?
5. We investigate a different algorithm to explain the characteristics that interfere with the defectiveness of software classes.

LITERATURE REVIEW

According to Novi Setiani etal. (2020) several automated test case generation techniques have been proposed to date, although the adoption of such technologies in industry is low. A major factor that has contributed to this low adoption rate is the difficulty experienced by the developer in terms of automatically reading and understanding test cases. For this reason, it is necessary to construct an understandable model of a test case to improve the generated test case.

H. Wu, C. Nie, J. Petke, Y. Jia, and M. Harman (2020), ‘‘told that an empirical comparison of combinatorial testing, random testing and adaptive random testing.

D. Honfi and Z. Micskei. (2019). Stated that Classifying Generated White-Box Tests: An Exploratory Study.

S. Scalabrino, G. Bavota, C. Vendome, M. Linares-Vasquez, D. Poshyvanyk, and R. Oliveto (2019) told that automatically assessing code understandability.

G. Grano, S. Scalabrino, H. C. Gall, and R. Oliveto (2018) tolad that an empirical investigation on the readability of manual and generated test cases.

H. Wu, C. Nie, J. Petke, Y. Jia, and M. Harman (2018) told that an empirical comparison of combinatorial testing, random testing and adaptive random testing.

According to A. Panichella, F. M. Kifetew, and P. Tonella (2017) the larger the program size, the greater the resources needed to write the class content. Automatically detect test cases at both the unit level, through black-box testing, using the random test method and whitebox testing, involving dynamic symbolic execution (DSE) and search-based software testing (SBST). Several methods of generating have been proposed.

RESEARCH METHODOLOGY

This research will be done in five stages. First, we have to extract the dataset to get the test case and developer metrics. We will be able to understand the test case by the developers' answers and the time required. Then, we will construct the dataset as a matrix for classification and regression algorithms. Last, we will perform a model evaluation using some measurements.

Feature Extraction

The first phase of this research is to extract the HONFI dataset. We use a subset of the dataset as the main result of using Hon. This dataset will originate from the experiment of classifying white box generated test cases involving 30 developers and 15 test cases. This dataset consists of two collections, set of test case files and results of experimental evaluation of test cases from the developer. Honorable generates the test cases under test using the intelligent tool for each tool. Evaluation of test cases to the developer is measured by the time required to understand the results of the test cases and the developer's answers. The first step in our methodology is extracting these generated test cases to obtain 20 test case metrics.

Data preparation

After 26 metrics are extracted from test case files and developer profiles, the next step is merging these two collections into the matrix m x n. The number of rows, the meter, denotes the count of the test case classification result made by the developer. Thirty developers assess 15 test cases, then M = 450 rows. While n is the number of matrices calculated from test cases and developers, where the total is 26.

Find the usefulness of a test case

In the previous model, the developer's understanding is measured using the actual and perceived understanding presented in the six understanding proxies. The six references are taken in the following context: The developer is given a piece of code, asked to read the code and answer the question: whether or not to understand the code. Last, they are given some questions related to the code. In terms of test case understanding, we use two proxies that derive from two aspects of test code comprehension: the accuracy of the developer's response in evaluating the test case evaluation, and the time required to understand the test code. There are two screens that can be extracted from Honfi's et al. Dataset:

1. Actual Binary Understandability (ABU)
2. Time Actual Understandability (TAU)

Classifier and Regression

To construct a model for predicting ABUs and TAUs, we use various regressions and regression options that have been defined in the literature to compare their performance. In particular, we use the classifier algorithm for modeling ABU with binary classes: (i) decision tree C.45 (j48), (ii) the Byers network, (iii) auxiliary vector machines (SMO algorithms), and (iv) Multilayer Perceptron Network. We use a regression algorithm for modeling TAU and NTAU with numerical classes: (i) random forests, (ii) linear regression (iii) auxiliary vector machines (SMO algorithms), and (iv) multilayer perceptron networks.
The ABU data were unbalanced, with 75% of the data presenting the correct answer from the respondent. We use the SMOTE filter on the training set to obtain a balanced model by generating artificial instances that represent incorrect answer data. It is important to have a compact subset feature, so we used principal component analysis to construct a set of values of linearly uncorrelated features. The ABU model is evaluated using AUC and F-measure for combined metrics, developer-related metrics, and test code metrics. The TAU model is interpreted using the Mean Absolute Error and Correlation Value because the approximate attribute is a numerical type. To trait and test the classification and regression models, a 10-cross validation technique is used. Its purpose is to avoid over fitting, the model is suitable only for nuances and cannot be generalized. The data is divided into ten partitions. Originally, nine partitions are used to build the model, and one partition is used to evaluate the model. Therefore, for each model created, it will be tested using data that is always different from training data.

Evaluation Metrics

To measure the performance of the classification model, we used the AUC and F-measure values. The AUC value represents the area under the ROC curve. The ROC curves described the performance of the model in classifying it into different threshold settings. The performance of the model is represented by real positive rate (sensitivity) values plotted on the y-axis and false-positive rate values plotted on the x-axis. AUC values are calculated using integral operation on RUC curves, whose values are in the range 0–1. The higher the AUC value, the better it is whether the model performs better in understanding the test case. F-measurement is one of the evaluation methods in classification models that mix recall and exact value. In a situation the value of recall and precision may have different weights. The F-measure introduces reciprocity between recall and precision, with the harmonic mean of recall and precision weighted

$$Fmeasure = \frac{2 \times Recall \times Precision}{Recal + Precision}$$

We used Pearson correlation and MAE (mean absolute error) in the equation to evaluate the performance of the prediction model. The MAE value represents the mean absolute error between the predicted value and the true value. The MAE is mathematically as follows:

$$MAE = \frac{1}{n} \sum_{i-1}^{n} |f_i - y_i|$$

f_i: predicted value
y_i: actual value
n: number of data.

RESULT ANALYSIS

The performance of SVFCS is assessed through measures based on Confusion Matrix. A confusion matrix (also called a contingency table) is a measurement table that relates the predicted results to the actual values of the dataset, and therefore, makes it possible to derive measures about the classifier's hits and misses. For a given model and set of examples, there are four possible outcomes associated with it. If the example is faulty (in the fault prediction approach) and it is predicted as faulty, it is counted as a true positive; If it is predicted as non-faulty, it is counted as a false negative.

If the instance is non-defective and is predicted as non-defective, it is counted as a true negative; If it is predicted as faulty, it is counted as a false positive. This two-by-two confusion matrix (for the binary class problem) forms the basis of many common metrics. Measurements based on the confusion matrix include accuracy, precision, recall and F-measure. These measures are widely used in the literature to evaluate the prediction results of a classifier. Next we told about all these measures.
Receiver operating characteristics graph is a well-known technique for assessing the performance of classifiers. It is a two dimensional graph in which the abscissa represents the probability of false alarm or cost and the coordinate represents the probability of detection or profit. It can handle both discrete and continuous classifiers. Discrete classifiers only produce class levels, while continuous classifiers produce a curve. Each discrete classifier generates a pair of PF and PD which represent a point on the ROC graph.
Some interesting points in the definition of the ROC curve include:
(a) The point (0, 0) indicates that it will never trigger a false alarm and will never issue a positive classification.
(b) Any point on the line drawn between (0, 0) to (1, 1) has no information.
(c) The point (0, 1) denotes the ideal situation. But this is never achieved by any classifier. So any classifier located closer to this point is always better.
Experiments were performed by dividing the total data set into training with 67% and testing with 33%. We repeated our experiments 10 more times on two well-known classifiers NB and SVM. The Iris and Pima Indians diabetes data sets are used for model validation. Our model on the iris data set gives 100% recall (detection probability also denoted as PD) and 0% false alarm rate (PF) with 100% accuracy, where on the Pima Indians diabetes data set it is 68.6 % and gives 29.5% PD and PF. which is better than the previous known results (60% and 19%[23]), except for the PF which should be lower. The validation results using these two datasets show that we can use this model in software defect prediction.

CONCLUSION

We introduced a new approach to software identification Comprehension concerns comprehension from a summary of the issue using vague methods and linguistic patterns. We generated a set of unambiguous rules from the dataset given in , and evaluated the performance of these rules on 1416 issues detected in Testopia and Tomcat 7. The results show that our model can achieve a high performance with an average accuracy of 98%. , between the two systems an accuracy of 93%, a recall of 86%, and a measure of 90%. Therefore, we conclude that a fuzzy set of rules can effectively identify a summary of related issues that are understandable. We believe that our model contributes to existing quality assessment and management approaches. SVM, FIS and genetic algorithm are well known methods and used for classification in engineering and every branch of science. For the first time in this work, we combined these three learner gains to obtain a better prediction model. The model's performance is checked against an open source data project, where previous studies [10] have been evaluated on historical data sets that are publicly available but could not simplify the test work in actual development due to technological changes. . This study needs to be expanded for validation purpose. Initially it works on binary classification problem, we are planning to generalize this model for multi class problem. We also plan to investigate the effect of support vectors on the performance of other classifiers.

BIBLIOGRAPHY

[1]. NOVI SETIAN, (Member, IEEE), RIDI FERDIANA1 , (Member, IEEE), AND RUDY HARTANTO (2020), Test Case Understandability Model, Received August 20, 2020, accepted August 30, 2020, date of publication September 9, 2020, date of current version September 24, 2020. Digital Object Identifier 10.1109/ACCESS.2020.3022876, VOLUME 8, 2020.

[2]. H. Wu, C. Nie, J. Petke, Y. Jia, and M. Harman, ‘‘An empirical comparison of combinatorial testing, random testing and adaptive random testing,’’ IEEE Trans. Softw. Eng., vol. 46, no. 3, pp. 302–320, Mar. 2020, doi: 10. 1109/TSE.2018.2852744.

[3]. D. Honfi and Z. Micskei. (2019). Classifying Generated White-Box Tests: An Exploratory Study. Accessed: Feb. 11, 2020. [Online]. Available: https://zenodo.org/record/2596044#.XkI4fhMzbUp

[4]. G. Fraser and A. Arcuri, ‘‘1600 faults in 100 projects: Automatically finding faults while achieving high coverage with EvoSuite,’’ Empirical Softw. Eng., vol. 20, no. 3, pp. 611–639, Jun. 2015, doi: 10.1007/s10664- 013-9288-2.

[5]. S. Scalabrino, G. Bavota, C. Vendome, M. Linares-Vasquez, D. Poshyvanyk, and R. Oliveto, ‘‘Automatically assessing code understandability,’’ IEEE Trans. Softw. Eng., early access, Feb. 25, 2019, doi: 10.1109/TSE.2019.2901468.

[6]. G. Grano, S. Scalabrino, H. C. Gall, and R. Oliveto, ‘‘An empirical investigation on the readability of manual and generated test cases,’’ in Proc. 26th Conf. Program Comprehension (ICPC), 2018, pp. 348–351, doi: 10. 1145/3196321.3196363.

[7]. H. Wu, C. Nie, J. Petke, Y. Jia, and M. Harman, ‘‘An empirical comparison of combinatorial testing, random testing and adaptive random testing,’’ IEEE Trans. Softw. Eng., vol. 46, no. 3, pp. 302–320, Mar. 2020, doi: 10. 1109/TSE.2018.2852744.

[8]. S. Scalabrino, M. Linares-Vasquez, D. Poshyvanyk, and R. Oliveto, ‘‘Improving code readability models with textual features,’’ in Proc. IEEE 24th Int. Conf. Program Comprehension (ICPC), May 2016, pp. 1–10, doi: 10.1109/ICPC.2016.7503707

[9]. A. Panichella, F. M. Kifetew, and P. Tonella, ‘‘Automated test case generation as a many-objective optimisation problem with dynamic selection of the targets,’’ IEEE Trans. Softw. Eng., vol. 44, no. 2, pp. 122–158, Feb. 2018, doi: 10.1109/TSE.2017.2663435.

[10]. S. Scalabrino, M. Linares-Vasquez, D. Poshyvanyk, and R. Oliveto, ‘‘Improving code readability models with textual features,’’ in Proc. IEEE 24th Int. Conf. Program Comprehension (ICPC), May 2016, pp. 1–10, doi: 10.1109/ICPC.2016.7503707

[11]. P. Ammann and J. Offutt, Introduction to Software Testing. Cambridge, U.K.: Cambridge Univ. Press, 2016, doi: 10.1017/9781316771273.

[12]. Jin-Cherng Lin and Kuo-Chiang Wu. Evaluation of software understandability based on fuzzy matrix. In FUZZ-IEEE, pages 887-892, 2008.

[13]. Jitender Kumar Chhabra, K. K. Aggarwal, and Yogesh Singh. Code and data spatial complexity: two important software understandability measures. Information & Software Technology, 45(8):539{546, 2003.

[14]. Jitender Kumar Chhabra, K. K. Aggarwal, and Yogesh Singh. Measurement of object-oriented software spatial complexity. Information & Software Technology, 46(10):689-699, 2004.

[15]. Maurice H. Halstead. Elements of Software Science. ISBN:0-444-00205-7. Amsterdam, 1977.

[16]. Seyyed Mohsen Jamali. Object oriented metrics. Department of Computer Engineering Sharif University of Technology, 2006.

[17]. Feng Jiang, Yuefei Sui, and Cungen Cao. A rough set approach to outlier detection. volume 37, pages 519–536. International Journal of General Systems, october 2008.

[18]. K.Shima, Y.Takemura, and K.Matsumoto. An approach to experimental evaluation of software understandability. Proceedings of the 2002 International Symposium on Empirical Software Engineering(ISESE’02), 2002.

[19]. Xiangjun LI and Fen RAO. An rough entropy based approach to outlier detection. Journal of Computational Information Systems, pages 10501–10508, 2012. Department of Computer science and Technology, Nanchang University,Nanchang 330031, China and College of Economy and Management, Nanchang University, Nanchang 330031, China.

[20]. Jin-Cherng Lin and Kuo-Chiang Wu. A model for measuring software understandabil- ity. In CIT, page 192, 2006.

[21]. Jin-Cherng Lin and Kuo-Chiang Wu. Evaluation of software understandability based on fuzzy matrix. In FUZZ-IEEE, pages 887–892, 2008.

[22]. Rajib Mall. Fundamentals of Software Engineering. Prentice Hall, 3rd edition, 2009.

[23]. Sanjay Misra and A. K. Misra. Evaluating cognitive complexity measure with weyuker properties. In IEEE ICCI, pages 103–108, 2004.