Decomposition of Classes on the Stability of Software Architecture

Harpreet Kaur and Kashish Sareen

***Abstract*—It has been explored if stability of software is related to the decomposing the class object are related to each other or not. Any part of the project code is suffering from the "Yoyo problem" with multiple issues related to readability of code, understandability of code as well as maintainability of code there is need to rethink, redesign, refactor these pieces of code . So as to find the best way to do is to simplify the inter relationship of class objects in such manner that code becomes concise with Liskov Substitution Principle with the help of decomposition of classes however this may lead to unknown or unwanted of class objects that may turn into stable, low, high, medium categories after the application of decomposition of class objects in such manner that code becomes concise and affecting the stability of overall application we are developing which may even lead to software erosion.**

***Index Terms*—Design stability, metrics, software architecture, dependency.**

I. Introduction

Object-oriented programming (OOP) to support software maintenance and reuse by introducing concepts like abstraction, encapsulation, aggregation, inheritance and polymorphism. However, years of experience have revealed that this support is not enough. Whenever a crosscutting concern needs to be changed, a developer has to make a lot of effort to localize the code that implements it. This may possibly require him to inspect many different modules, since the code may be scattered across several of them [1].

Object-oriented programming address (at least) the three major software engineering goals shown in this Fig. 1.

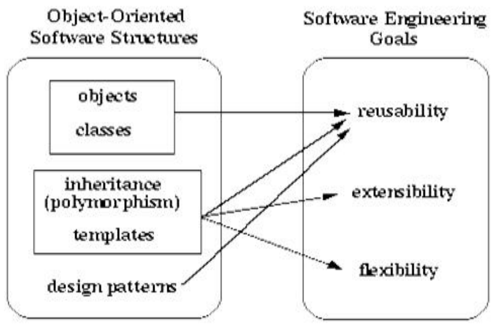
Object-Oriented Software Structures
Software Engineering Goals
objects
classes
inheritance (polymorphism)
templates
design patterns
reusability
extensibility
flexibility

Fig. 1. Software engineering goals.

Manuscript received January 10, 2014; revised April 17, 2014.

Harpreet Kaur is with the Computer Engg Deptt. at K.C. Group of Institutes, Nawanshahr, India (e-mail: justpreeti@gmail.com).

Kashish Sareen is with the Electronics and Communication Engineering Department in K.C. College of Engineering & I.T., Nawanshahr, India (e-mail: kashishsareen@yahoo.co.in).

An essential problem with traditional programming paradigms is of the dominant decomposition. No matter how well a software system is decomposed into modules, there will always be concerns (typically non-functional ones) whose code cuts across the chosen decomposition. The implementation of these crosscutting concerns will spread across different modules, which has a negative impact on maintainability, stability and reusability [1].

One of the most important difficulties for solving large-scale real-world problems is how to divide a problem into smaller and simpler sub-problems; how to assign a network module to learn each of the sub-problems; and how to recombine the individual modules into a solution to the original problem [4]. By decomposing (separation of a substance into simpler substances or basic elements) means making more and more classes. Decomposition is done by adding inheritance to the existing class. Inheritance is done up to single level only. A problem is broken down into a set of sub-problems according to the inherent class relations among training data [2]. But after decomposition stability is affected. Stability of software is ability of software to remain unchanged over time under stated or reasonably expected conditions. A stable software release is so named because it is unchanging [6].

Software that is intended for the public to use is usually "stable". It is released, and following the release no new features are added apart from the odd bug fix. To get new functionality users eventually need to upgrade to the next version. Any problems with the software (unless they can easily be fixed with a bug fix update) are "known" problems, and the software vendor does not need to keep track of more than one variation of these problems for any given version. Another meaning of stable exists in common use, where people take it to mean "working reliably" or "solid". That is, people refer to software that runs consistently without crashing as stable.

II. Research Gap

After conducting systematic literature survey and from the possible study material related to the decomposition of classes and its impact on stability of the application as such we have found that limited work has been done in this area , in fact, decomposition of the classes have been carried out to find it affects the overall maintainability index of each class in question, but not in terms of the stability of the software. Moreover, limited work has been done to automate the process of finding the degrees of stability of software using any machine algorithm. Hence, in current context of research, there are ample options to improve the work in this field by taking the advantage of machine learning algorithms like SVM, Na ïve Bayes and logic.

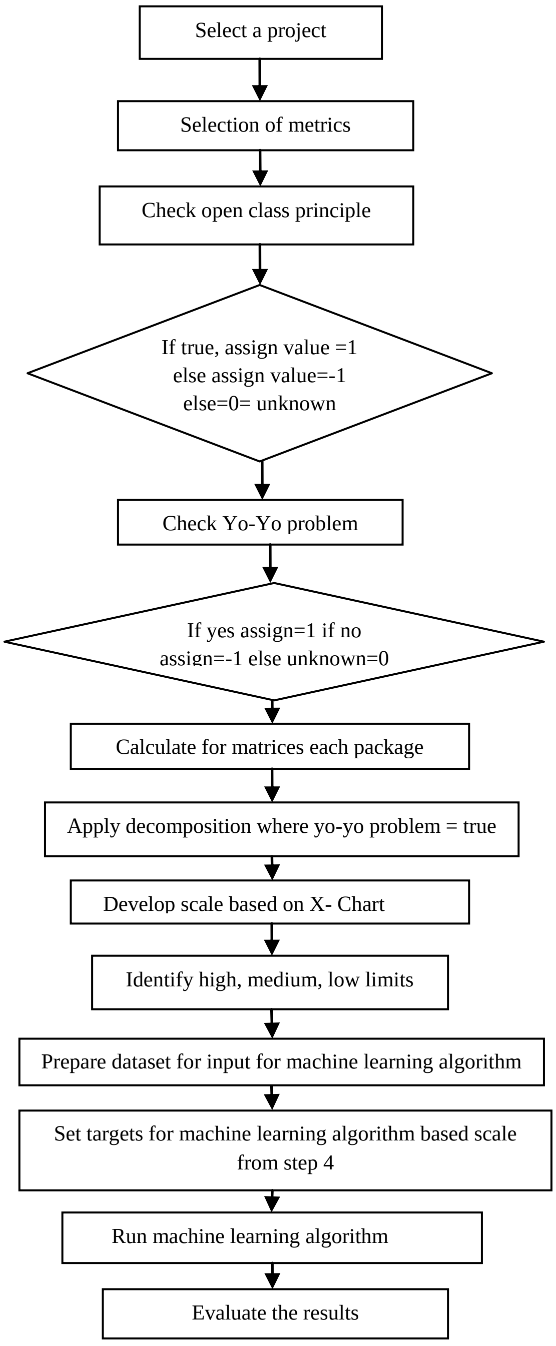
Select a project
Selection of metrics
Check open class principle
If true, assign value =1
else assign value=-1
else=0= unknown
Check Yo-Yo problem
If yes assign=1 if no
assign=-1 else unknown=0
Calculate for matrices each package
Apply decomposition where yo-yo problem = true
Develop scale based on X- Chart
Identify high, medium, low limits
Prepare dataset for input for machine learning algorithm
Set targets for machine learning algorithm based scale
from step 4
Run machine learning algorithm
Evaluate the results

Fig. 2. Calculating the Stability of Software.

III. Objectives

1) Develop a representative datasets of projects in which stability has been challenging.
2) Develop an ordinal scale to measure stability of software based on metrics and descriptive statistics of the value of metrics.
3) Find most suitable machine learning algorithm for automating the algorithm for find the degree of stability of software.
4) Evaluate the performance of the system in terms of Recall and Process values.

IV. Methodology Steps

Methodology for how to calculating the stability of software in Fig. 2.

V. Results

The results shown below are based on dataset that are below and after application of decomposition. The performance of the different classifiers was compared amongst each other – Logistic, RBF, SVM, Na ïve Bayes in terms of Kappa Statistics, Precision, Recall, Root Mean Square Error. The detailed results are discussed in the following sections.

A. Kappa Statistics

Kappa statistics is a measure of inter-rater agreement or inter annotator agreement for qualitative (categorical) items. Landis and Koch characterized values < 0 as indicating no agreement and 0–0.20 as slight, 0.21–0.40 as fair, 0.41–0.60 as moderate, 0.61–0.80 as substantial, and 0.81–1 as almost perfect agreement.

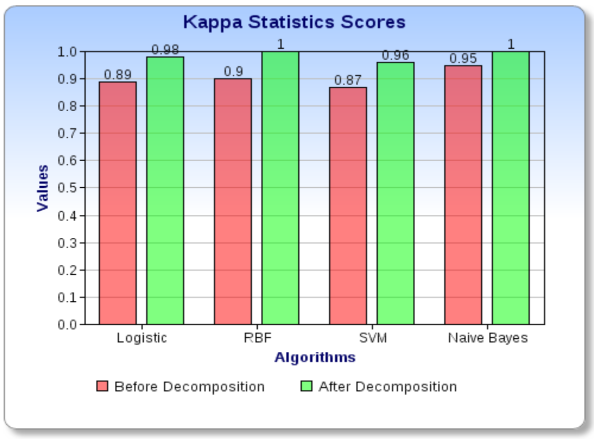

Fig. 3. Kappa Statistics a comparative view.

B. Precision

PRECISION is the ratio of the number of relevant records retrieved to the total number of irrelevant and relevant records retrieved. It is usually expressed as a percentage. Precision $= \frac{A}{A+C} \times 100$, A= Number of relevant records retrieved, C= Number of irrelevant records retrieved. Closer is the value of $\frac{A}{A+C}$ to 1 higher the precision. Value closer to 1 indicates that no irrelevant records are being retrieved.

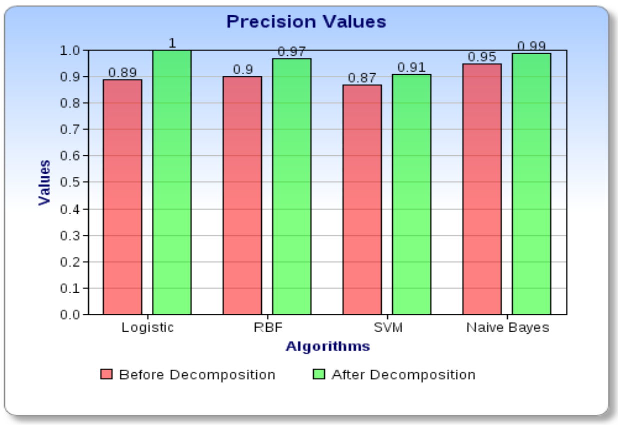

Fig. 4. Comparison of precision.

C. Recall

RECALL is the ratio of the number of relevant records retrieved to the total number of relevant records in the database. It is usually expressed as a percentage.

Recall = $\frac{A}{B} \times 100$,

A= Number of relevant records retrieved,

B= Number of relevant records not retrieved Again closer the value $\frac{A}{B}$ to 1 higher the recall.

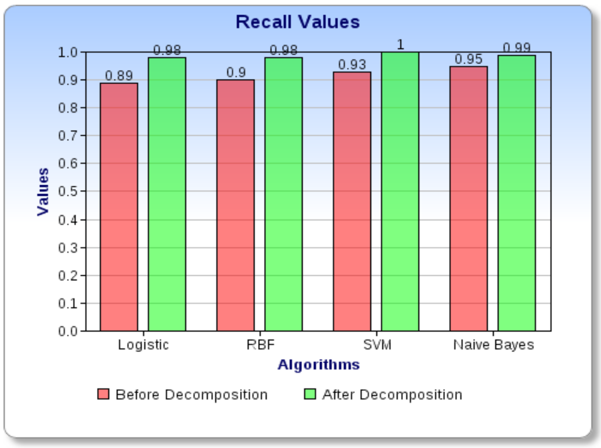

Fig. 5. Recall values for different algorithms.

D. Root Mean Squared Error

The Root Mean Square Error (RMSE) is a frequently used measure of the difference between values predicted by a model and the values actually observed from the environment that is being modelled. The RMSE of a model prediction with respect to the estimated variable X_{model} is defined as the square root of the mean squared error:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(X_{obs,i} - X_{model,i})^2}{n}}$$

where X_{obs} is observed values and X_{model} is modelled values at time/place *i*.

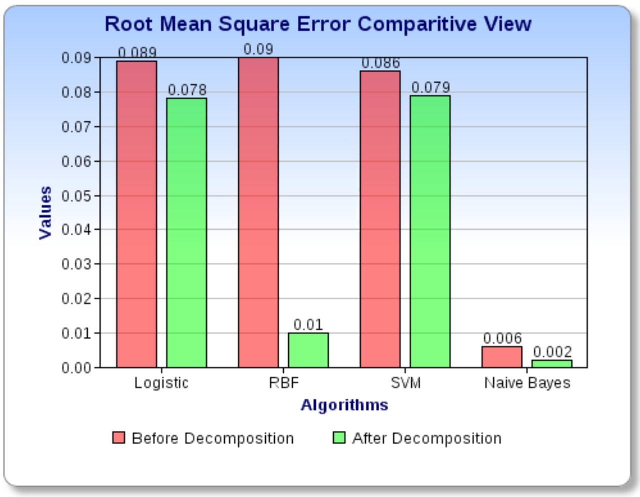

Fig. 6. Root mean squared error for different classifiers.

E. False Positive Rate

The *false positive rate* (*FP*) is the proportion of negatives cases that were incorrectly classified as positive, as calculated using the equation:

$$FP = \frac{b}{a+b}$$

b = incorrect number of predictions that an instance is negative
a = correct number of predictions that an instance is negative

High value of false positive rate indicates that the algorithm is making large number of incorrect predictions and hence it is not reliable.

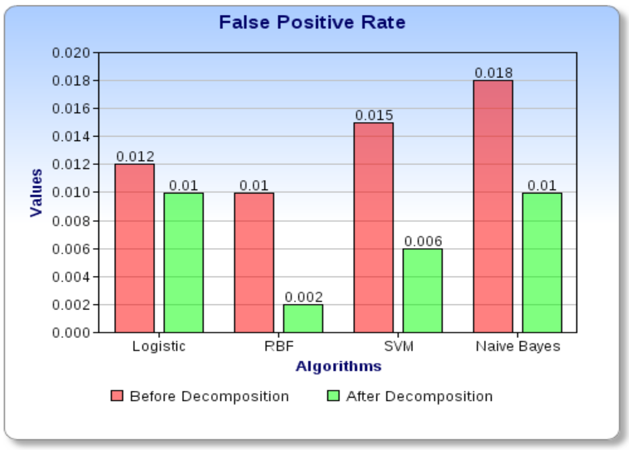

Fig. 6. A comparison of False Positive Rate for different algorithms.

Fig. 7 indicates that RBF and SVM classifiers exhibit low false positive rate. RBF is showing the lowest FPR. FPR in case of Logistic and Naïve Bayes algorithm are highest. Low false positive rate is an indication of high level of accuracy.

F. True Positive Rate

The *true* positive *rate* (*TP*) is the proportion of positive cases that were correctly identified, as calculated using the equation:

$$TP = \frac{d}{c+d}$$

d is the number of **correct** predictions that an instance is **positive**. *c* is the number of **incorrect** of predictions that an instance **negative**, A high positive rate indicates that the classifier is capable of prediction which is very near to the specified criteria.

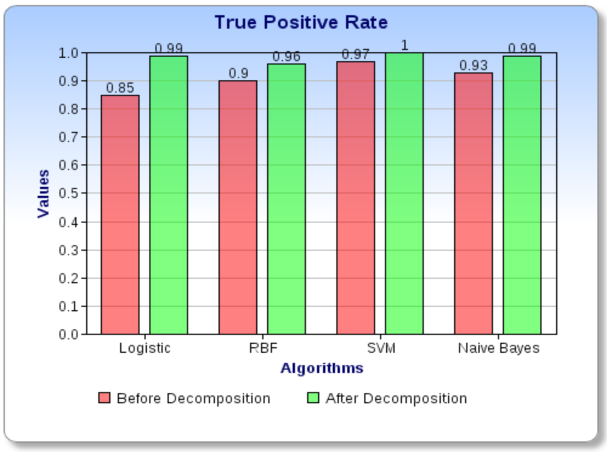

Fig. 7. True Positive rate for different classifiers.

Fig. 7 reveals that for SVM the True positive rate is maximum which can be due to same value of c(number of incorrect predictions that an instance negative) & d(number of correct predictions that an instance is positive. By which the ratio is 1 and for Logistic, NB and RBF the value is less than 1 which can be due to large value of term in the denominator (*c*+*d*).

VI. Conclusion

Summarization of the results of the study is shown in Table I.

TABLE I: Summary of the Result

Material/ Algorithm	LR Before	LR After	RBF Before	RBF After	SVM Before	SVM After	NB Before	NB After
KS	0.89	0.9878	0.9	1	0.87	0.9678	0.95	1
RMSE	0.089	0.0781	0.09	0.1	0.086	0.0791	0.006	0.002
Precisi on	0.89	1	0.9	0.97	0.87	0.91	0.95	0.99
Recall	.89	0.98	0.9	0.98	0.93	1	0.95	0.99
FP	0.012	0.01	0.01	0.002	0.015	0.006	0.018	0.01
TP	0.85	0.99	0.9	0.96	0.97	1	0.93	0.99

The results of the study indicate that Naïve Bayes classifier is better for detecting stability of software. Kappa value, Precision and Recall all these parameters have values nearly equal to 1. A value of 1 effectively means successful prediction of true positive rates. Unity value of Kappa statistics, and Precision (accuracy) further reinforces the effectiveness of Naïve Bayes classifier. Recall value is an indicator of the ability of the algorithm to reproduce the same results over a period of time. Recall value of 0.99 for Bayes, Network indicate that the algorithm when subjected to same conditions over different period of times gives the same results, indicating high reliability of the algorithm.

It is clear from the results that after the process of implementing decomposition the dataset become more discriminate and difference between each class increased. It become for all algorithms to; identify the boundary/hyper plane more easily, thus the accuracy increase after decomposition. This may be attributed to the fact that the number of classes/components for which accuracy is calculated is increased; which leads to change in previous metric value and new value to a greater value.

VII. Future Scope

To begin with research we explored how the *Liskov Substitution Principle* and problem of "Yoyo" need to be considered for having a stable, mature and reliable software. For future scope we suggest more metrics that have deep causal relation with respect to the stability of the software may be developed and validated for understanding the effect not only the stability but readability of the software/application we focus on artificial intelligence for automating the process

References

[1] A. Przybyłek, "Systems Evolution and Software Reuse in Object-Oriented Programming and Aspect-Oriented Programming," University of Gdansk.

[2] A. Chhikara, R. S. Chhillar, R. S. Chhillar, "Evaluating the impact of different types of inheritance on the object oriented software metrics," in *Proc. the International Journal of Enterprise Computing and Business Systems*, July 2, 2011.

[3] B. Boehm and K. J. Sullivan, "Software Economics: A Roadmap. In: Finkelstein," The Future of Software Engineering, 2000.

[4] B.-L. Lu and M. Ito, " Task decomposition and module combination based on class relations: a modular neural network for pattern classification," *Transactions on Neural Networks*, vol. 10, no. 5, September 1999

[5] G.-Y. Tang and H.-W. Xuan, "Research on Measurement of Software Package Dependency based on Component," in *Proc. the Journal of Software*, vol. 7, no. 9, September 2012.

[6] H. Mannaert, J. Verelst, and Kris, "Towards evolvable software architectures based on systems theoretic stability," Wiley Online Library, 2011.

[7] M. Yadav, P. Baniya, and G. Wayal, "Comparison Between Inheritance & interface UML Design Through the Coupling Metrics," in *Proc. the International Journal of Engineering and Advanced Technology (IJEAT)*, June 5, 2012.

[8] P. Greenwood, T. Bartolomei, E. Figueiredo, M. Dosea, A. Garcia, N. Cacho, C. Sant'Anna, S. Soares, P. Borba, and A. Rashid, *On the Impact of Aspectual Decompositions on Design Stability: An Empirical Study*.

[9] S. Yau and J. S. Collofello, "Design stability measures for software maintenance," *IEEE Trans. Softw. Eng.*, vol. 11, no. 9, pp. 849–856, 1985.

[10] S. Chidamber and C. Kemerer, "A metrics suite for object oriented design," *IEEE Trans. on Soft. Eng.*, vol. 20, no. 6, pp. 476–493, 1994.

Harpreet Kaur was born on 4th March 1986 in India. She received her B.Tech degree in computer science engineering 2007 from IET Bhaddal Ropar, India respectively under Punjab Technical University, and pursuing M.tech in information technology from Guru Nanak Dev Engineering College, India. She is currently working as a senior lecturer in computer engg deptt. at K.C. Group of Institutes. Nawanshahr India. Her research interests include quality of service issues in software engineering.

Kashish Sareen was born in India. He received his B.Tech and M.Tech degrees in electronics and communication Engg. under Punjab Technical University, India. He got his B.TECH degree in 2008 from K.C. College of Engineering & I.T., Nawanshahr India and M.TECH from Guru Nanak Dev Engineering College, India. He is currently working as an assistant professor and head of Electronics and Communication Engineering Department in K.C. College of Engineering & I.T., Nawanshahr India. His research interests include quality of service issues in ad hoc wireless and sensor networks.