Supplementing Object-Oriented Software Change Impact Analysis with Fault-proneness Prediction

Bassey Isong, Ohaeri Ifeoma
Computer Science Department
Material Science and Innovation
North-West University, Mafikeng, South Africa
bassey.isong@nwu.ac.za, oh.ifeoma@yahoo.com

Munienge Mbodila
Computer Science and Info. Systems Department
University of Venda
Thohoyandou, South Africa
Munienge.mbodila@univen.ac.za

***Abstract*—Software changes are inevitable during maintenance, Object-oriented software (OOS) in particular. For change not to be performed in the "dark", software change impact analysis (SCIA) is used. However, due to the exponential growth in the size and complexity of OOS, classes are not without faults and the existing SCIA techniques only predict change impact set. This means that a change implemented on a faulty class could increase the likelihood for software failure. To avoid this issue, maintenance has to incorporate both change impact and fault-proneness (FP) prediction. Therefore, this paper proposes an extended approach for SCIA that integrates both activities. The goal is to assist software engineers with the necessary information of focusing verification and validation activities on the high risk components that would probably cause severe failures which in turn can boost maintenance and testing efficiency. This study built a model for predicting FP using software metrics and faults data from NASA data set in the public domain. The results obtained were analyzed and presented. Additionally, a class change recommender (CCRecommender) tool was developed to assist in computing the risks associated with making change to any component in the impact set.**

Keywords—Change impact, FP, OOS, metrics, prediction

I. INTRODUCTION

In the world of software development today, due to the exponential upsurge in the size and complexity of software applications as well as the software failure criticality, ensuring high quality in large software systems during development has become more and more difficult and a time-consuming task [4][5][15]. As a result, software deliverables with serious faults that could lead to failure in the field are produced [1][3][4]. To assure high-quality in the software, faults have to be found and removed. Though, testing, inspection, and walkthrough are designed for this task, testing activity waist time and resources [8]. In software engineering, a cost-effective way of achieving high-quality is by predicting fault-proneness (FP) early in the software using software metrics. The importance is to provide essential information that can be used to focus verification and validation efforts on the affected components and boost maintenance and testing efficiency.

In addition, maintaining large systems today especially object-oriented software (OOS) has become a difficult task. Change is an indispensable property of any software that is necessary to keep the system alive and is inevitable during maintenance [9][10]. However, just like change in our society, software changes are not straight forward. Irrespective of the size, a change can cause some unpredictable effects on other components of the system [10]. In the perspective of OOS, we can say that a change carries some possibility of failure that could manifest either during testing or in the field. The failures could be due to either: (i) negligence of the existence of dependences between the OOS components, (ii) process activities such as the frequency of changes, number of developers performing the changes, their maintenance experiences, the file ages, etc. or (iii) faulty software components such as classes, methods and fields. The first case can be minimized or eliminated via effective software change impact analysis (SCIA) activities while the later can't be eliminated with SCIA activities alone. This calls for an effective approach to prevent severe software failure when changes are made. As classes are the basic units of analysis in OOS, their quality is critical to the overall quality of the software system. Nevertheless, OOS classes are not exception in terms of FP as reported in several empirical studies in the literature [1][3][4][8][11][12]. FP is the likelihood that a software component has at least one fault [8]. Since faults may lead to failure in an executable product, we believed that changing a faulty class without the prior knowledge of such faults could compound the risk of software failure. To guard against software failure due to maintenance in OOS, this study then proposes an approach that predict faulty classes in the impact set after SCIA activities before actual changes are made. By identifying faulty classes early would allow mitigating actions to be directed to the high risk components that could possibly cause field failures when changes are made.

This study constructed a model for predicting the FP of OOS components affected by change. We used data from the industry, KC1 NASA dataset [1][4][14] to establish FP and OO design/size metrics relationships. The metrics used are Chidamber and Kamerer (1994) metric suite [20] and software lines of code (SLOC) as well the faults data. Logistic regression (LR) was used to construct the model and the results obtained were analyzed and presented. In addition, a tool called class change recommender (CCRecommender) was developed to assist software engineers to quantify the risk associated with making change if any in the impact set. The tool is designed to assist them to identify fault-prone classes so that resources and

978-1-5090-0806-3/16/$31.00 copyright 2016 IEEE

verification efforts can be concentrated on the identified high-risk classes before actual changes are made.

The rest of the paper is organized as follows: Section II is the background information, Section III presents the extended SCIA framework and Section IV discusses the FP prediction model. Moreover, Section V is the methodology employed, Section VI is the results analysis while Section VII is a discussion on the model and Section VIII is the model application to software maintenance. Section IX is the paper conclusions.

II. Background Information

Software maintenance phase constitute one of the most key phases of software life cycle and has been tagged the most costly and difficult phase [17]. Studies have shown that about 70% or more of the total software life-cycle costs are consumed by software maintenance [18]. Software change is a fundamental operation to keep software alive and is in fulfilment of one of the laws of program evolution which advocate for continuing change. Changes in software over time are vital to cope with change requests such as add new requirements, fix faults, system enhancements and meet the changing needs of the customer [10][18] and so on. Maintenance is importance for the fact that several software organizations have invested whooping sums of money on their systems today. Thus, such systems have to be effectively maintained instead of developing new systems. However, changes can produce certain undesirable effects that could yield inconsistency on other components of the software [9][10][19]. To preserve the quality of the software, SCIA is employed. SCIA is a technique used to determine which software component will be truly affected by a change request or likely to be changed when a component is changed [10].

During the course of SCIA, the input is the request for change proposed over time. These requests coupled with the source code are analyzed and by applying SCIA technique, impact set is discovered. Several SCIA methods exist which are categorized into static analysis methods and the dynamic analysis methods [10][18]. Taking the static analysis methods into consideration, there are several approaches that exist in the standpoint of OOS. These approaches predict change impact set only. Unfortunately, OOS classes are not fault-free. OO classes are known to be faulty as a result of flaws in design and code complexity, though faults are only known to be present in a few system components [6][7]. In this case, performing a change on a class considered to be faulty could be risky and costly. Since testing activity is known to consume significant amount of time and resources with regards to software size and complexity, achieving high quality during maintenance can be effective via predicting the FP of the affected classes in the impact set before final changes are made. That is, predicting the most critical components of the software systems where faults are likely to occur early in order to appropriately allocate resources for detecting and fixing them [8][11]. This is vital to guard against unintended risks posed by faulty components affected by change request that are likely to cause systems failure in the field. In addition, the earlier faults are identified, the lower the costs of fixing and the higher the quality of the software deliverables [1][5][16].

In software engineering, software fault prediction constitute one of the most efficient techniques to achieve high quality. Fault prediction depends on past software release measures and its faults data to identify the fault-prone components (i.e. classes) for the next release [8][11]. Today, several techniques have been developed such as neural network, statistical, machine learning techniques and so on [13] to predict faults in a software component. These techniques uses software metrics to evaluate the quality of the software product. Several OO metrics have been proposed and are used in the evaluation of both its OO design and codes quality. Moreover, many studies have empirically validated the relationship between OOS measures and FP [13]. The validation is aimed at ensuring that the OO measures continue to remain relevant in the evaluation of software quality and the early prediction of high risk components of the software [11].

III. The Extended SCIA Framework

In this section, we present the contribution of this paper to the field of software maintenance - *extended SCIA framework*. Extended SCIA framework is an approach that incorporate both change impact prediction and FP prediction to enhance existing SCIA activity. FP prediction is considered important because existing OO SCIA techniques only predicts impact set without considering the FP of OO classes. Moreover, though existing version control system maintained during configuration management take control of all OOS dependencies thereby putting complexity under check, there are cases where such dependencies are neglected or the task of maintaining the dependencies might be time consuming. In the presence of such cases, there is the likelihood that the system under maintenance may have some faults that are hard to detect or identify the impact associated with a change. Consequently, if changes are made to such classes that are faulty, it could however increase the risk for software failure in the field. By prediction FP of the impact set will allow faulty classes to be identified and take decisions early before actual changes are implemented.

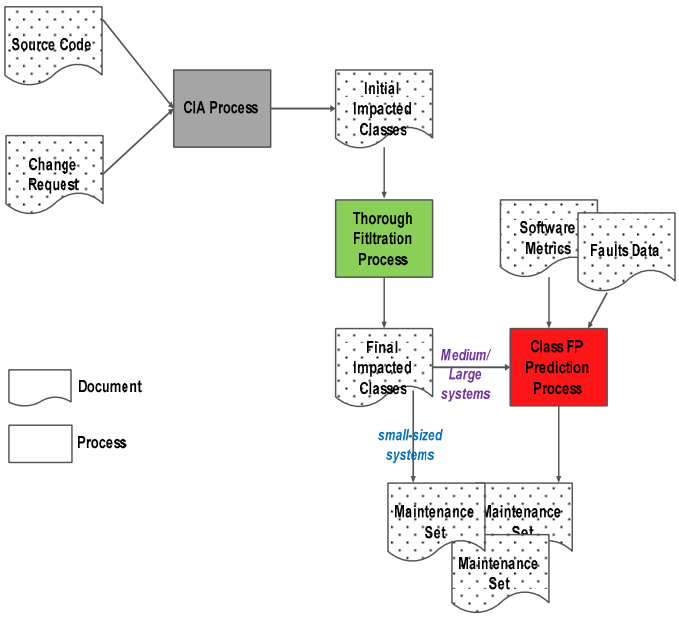

Fig. 1. Extended SCIA framework

Figure 1 presents the extended SCIA framework and it shows that, before a change request is implemented, faulty classes that are likely to be faulty or lead to failure in the field should first be identified and fixed. We consider this vital to avoiding risky software changes. The approach shown in Figure 1 involve three main phases: *impact prediction*, *FP prediction* and *change implementation*. There are discussed as follows:

A. Impact prediction

As shown in Figure 1, during the course of a maintenance task, the approach will assume a normal approach for conducting SCIA. That is, any static impact analysis technique for OOS can be applied while taking the change type and the dependencies between impacted components into account. The objective is to assist in determining which OOS components in the original software systems are truly affected by the change request or yield inconsistencies in the system. The resultant output will therefore, be the correct impact set that are the real candidate for maintenance which in turn, will be utilized as input in the next phase.

B. Fault-proneness Prediction

After completing the main SCIA activity with output as the impact set, the next activity would have been to perform the actual changes. However, in this proposed approach, changes are performed after predicting the FP of OOS components in the impact set. The basis of predicting FP at this phase is to determine which among the affected components may lead to software failure or increase the risk of failures if changes are committed. As OO classes are complex and not without faults, identifying fault-prone components in the impact set is important. However, predicting FP is dependent on the size (i.e. small, medium and large) of the system and the availability of fault data from the earlier releases. For instance, for small sized systems, FP prediction can be avoided. Consequently, only the impact set that can be predicted. On the other hand, for a medium-sized or large-scale systems, FP prediction is important. In this case, faults data from previous release can be used coupled with snapshot of software measures from current version and past change histories will be used. To achieve this, suitable fault prediction model can be utilized. The importance of identifying or detecting faulty OOS components early before actual change implementation is aimed at reducing maintenance efforts, costs and risky changes while assuring high-quality software products. In addition, regression testing and other decision making activities can be facilitated.

C. Change Implementation

Before change can be implemented, decisions has to be made by taken the impact set and their FP into consideration. Based on the assessment made, decisions can be reached as to either accept the change, reject the change if it associated with deteriorating effects or consider further change plan such as refactoring to improve the system. Additionally, plans will involve channeling verification and validation activities on the high risk components in order to reduce or eliminate the risk of software failure when changes are made.

IV. Fault-Proneness Prediction Model

When constructing a FP prediction model, several decisions have to made about which the variables to be used, the statistical technique, the evaluation method and the evaluation criteria [11][22]. In this paper, to construct a prediction model, we first identify the variables: dependent and independent and then proceed to describing the model construction techniques and its evaluation.

A. Variables and Fault Data

1) Dependent and independent variables: This paper is centered on building a fault prediction model that will be used to improve the quality of OOS under maintenance. In the model, the dependent variable is the FP which is the faults data collected from previous release of the software system. The collected faults data are then used to evaluate whether OO design measures are useful for predicting the likelihood of classes being faulty. Moreover, the independent variables are OO design metrics and size metric. We utilizes Chidamber and Kamerer (1994) or CK metric suite [20] which are weighted methods per class (WMC), coupling between objects (CBO), response for a class (RFC), lack of cohesion of methods (LCOM), depth of inheretance (DIT), numbeer of childrence (NOC) and software lines of code (SLOC). The choice of these metrics is based on the fact that they have been empirically validated and re-validated several times both in the industry and the academia for being associated with OOS class FP [13].

2) Fault data description: To construct the prediction model, this study reused public domain data set KC1 collected from the NASA Metrics Data Program [14]. The collection and validation of the data set was performed by Metrics Data Program and stored in the NASA data repository. In addition, the KC1 dataset was gathered from a storage management system developed in C++ programming language used for receiving and processing ground data. For more information, visit [1][4][14]. The system has 145 classes with 2,107 methods and 40 KSLOC. Though the KC1 offers both static software metrics at both class and methods levels, only static metrics at the class-level are of interest in this study. Consequently, the values of 7 metrics were computed: CK and SLOC metrics at the class-level for analysis. Fault data were collected by associating methods to classes as well as their reported faults. Out of the 145 classes, 60 classes were found to be faulty while 85 are not faulty. A class is faulty if at least one fault is traced to it, otherwise, not faulty.

V. Methodology

This section presents the methodology used. We employed descriptive statistics in the analysis of the metrics and LR in the construction of the model.

1) Descriptive Statistics: Descriptive statistics play an important part in analyzing and quantitatively describing the main features of the data collected. It assist in reducing empirical data to a form that can be read easily, draw conclusion and further analysis. The important statistics measures used in this study for comparing the different metrics of interest are the

minimum, maximum, mean, median, and standard deviation as shown in Table I.

TABLE I. DESCRIPTIVE STATISTICS OF METRICS

Metrics	N	Min	Max	Mean	Std. Dev.
CBO	145	0	24	8.32	4.270
LCOM	145	0	100	68.72	18.362
NOC	145	0	5	.21	.747
RFC	145	0	222	34.38	26.493
WMC	145	0	100	17.42	14.597
DIT	145	0	6	1.00	1.149
SLOC	145	0	2313	211.25	190.033
Fault Data	145	0	101	4.61	10.841

In Table I, class size is measured with respect to lines of code that vary between 0 and 2313. SLOC has the highest mean indicating the largest measure per class in the KC1 data set. Also, LCOM measure has higher values when compared to CBO, indicating good quality in the design. Furthermore, DIT and NOC values are very low, indicating there were not much utilized in all the systems studied.

2) Binary Logistic Regression: Binary LR is one of the most widely used statistical techniques to model the relationship between certain internal products attributes and their external quality attributes such as FP. LR is the best statistical technique for building a prediction model [13]. It predicts the probability for an event to occur such as fault prediction and is used in constructing prediction model that utilized a dependent variable, Y as binary. That is, it can take on only one of two different values (0 and 1). Y = 1 indicates the class is faulty and Y = 0 indicates a non-faulty class [8][11]. In the model, Y is the measure of FP of a class. Suppose that $X_1, X_2, X_3,..., X_n$ are the independent variables and Prob(Y = 1|$x_1, x_2, x_3 . . ., x_n$) represents the probability that Y = 1 when $X_1 = x_1, X_2 = x_2, X_3 = x_3 . . .$, and $X_n = x_n$. Then, LR model assumes that Prob(Y = 1|$x_1, x_2, x_3 . . ., x_n$) is connected to $x_1, x_2, x_3 . . ., x_n$ as shown in equation (1). Thus, the general format of BLR model is given by:

$$Prob\left(\mathrm{Y}=1|\mathrm{x}_1,\mathrm{x}_2,\mathrm{x}_3\ldots,\mathrm{x}_n\right)=\frac{e^{\omega\ +\lambda_1x_1+\lambda_2x_2+\cdots+\lambda_nx_n}}{1+e^{e^{\omega\ +\lambda_1x_1+\lambda_2x_2+\cdots+\lambda_nx_n}}}\ldots1$$

Where Prob is the conditional probability which indicates the likelihood that a class has a fault. X_i is the independent variable which are the class's change history and structural properties and ω and λ_is parameters (where i=1, 2,..,n) are the estimated regression coefficients which are obtained through the maximum log-likelihood.

In this study, multivariate LR and univariate LR are employed. Multivariate LR is used to build the prediction model for classes FP. In this case, several metrics considered to be related to FP based on univariate and correlation analysis are used in combination to predict FP of classes. In equation (1), the variables $x_1, x_2, x_3 . . ., x_n$ are the predictors. On the other hand, the univariate LR model as shown in equation (2) is a distinct form of multivariate LR involving only a single independent variable, X. Univariate LR is used to model the association between the dependent and the individual independent variables in order to confirm their significance with respect to FP. The format for univariate LR is given by:

$$Prob(Y=1|x)=\frac{e^{\omega+\lambda x}}{1+e^{\omega+\lambda x}}\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots2$$

Other reported statistics used in this study are the estimated regression coefficients (ω and λ_is parameters), statistical significance (ρ(sig)), R-square Statistics (R^2), odds ratio (Exp(λ)), maximum likelihood estimation. For model evaluation, we employed sensitivity, specificity and accuracy.

VI. RESULTS ANALYSIS

This section present the result of the binary LR analysis. It begins with the computation of the univariate LR followed by the correlation analysis and lastly, the multivariate LR. The importance is to select optimum subset of metrics that will be used for constructing the class FP prediction model.

A. Univariate LR Analysis

Results of the univariate LR is presented in Table II. The statistics reported are the regression coefficient (λ), the statistical significance ρ- value, odds ratio (Exp(λ)), and R^2 of individual measure. Metrics were selected based on positive λ, ρ-value ≤ 0.05 and Exp(λ) > 1. A cutoff value of 0.5 was used for the classification. Based on the selection criteria, NOC metric has no significant relationship with FP while LCOM metric appears negative in its coefficient and its odds ratio is less than 1. Consequently, LCOM metric is inversely related to FP. Accordingly, metric with the highest value of R^2 is the CBO, signifying the best predictor followed by WMC metric. In addition, from the results obtained, R^2 values are considered more important than the ρ-values as they indicate the correlation strength. Also, CBO, RFC, and WMC are found to be more significant with respect to their R^2 value joined by SLOC and DIT. Thus, CBO is the most effective at that level of significance used while LCOM and NOC are dropped in further analysis.

TABLE II. ULR RESULTS

Metrics	λ	ρ-value	Exp (λ)	R2
CBO	2.040	0.000	7.688	0.649
LCOM	-0.091	0.000	0.913	0.290
NOC	0.009	0.969	1.009	0.000
RFC	0.127	0.000	1.135	0.506
WMC	0.171	0.000	1.187	0.443
DIT	0.585	0.002	1.794	0.081
SLOC	0.006	0.010	1.007	0.060

B. Correlation Analysis

Correlation analysis is used to discover relationship between individual software metric and the fault data - FP. The Spearman's rank correlation among metrics is presented in Table III. The correlation coefficient value was based on Hopkins [23] classification. In the classification we adopted, the threshold value is 0.5 and the significance level is ρ ≤ 0.01 level. In the results captured in Table III, WMC, DIT, SLOC are related to RFC while RFC, WMC and CBO are related to SLOC.

TABLE III. CORRELATION RESULTS

Metrics	CBO	RFC	WMC	DIT	SLOC
CBO	1				
RFC	.691**	1			
WMC	.619**	.529**	1		
DIT	.255**	.152	.367**	1	
SLOC	.171*	.312**	.110	.064	1

**. Correlation is significant at the 0.01 level (2-tailed).
*. Correlation is significant at the 0.05 level (2-tailed).

Moreover, the results indicates that the metrics are not completely independent or redundant with each other, proving they are significantly good for fault prediction. However, the correlations among CBO, RFC, and WMC and between RFC and WMC are very strong. Consequently, CBO, RFC, WMC, DIT and SLOC are selected to be included in the MLR model.

C. Multivariate LR Results

Multivariate LR analysis is aimed at selecting metric subsets that yield the optimum classification in the model. To achieve this, we used a forward stepwise procedure [24] and promising results were obtained in terms of R^2 and log likelihood statistic. The classifier at threshold = 0.5 was used as cutoff value, and all the 145 classes were used as the train data. Due to the multi-collinearity effect, we have three sets of predictors as potential metrics (M_1, M_2 and M_3) for the model construction. Table IV captured the regression coefficient (λ), statistical significance (ρ-value), odds ratio (Exp(λ)), R^2, the -2 Log likelihood and the constant (ω) for metrics included in the model, M_3. Moreover, Table V captured the results of the classification and Table VI presents the specificity, accuracy and sensitivity of the models.

TABLE IV. MLR RESULTS FOR M_3

M3	λ	ρ-value	Exp (λ)
CBO	2.938	0.005	18.878
RFC	0.200	0.027	1.221
DIT	2.214	0.021	9.152
R^2	0.709		
- 2 Log likelihood	17.605		
Constant (ω)	- 37.124		

TABLE V. CLASSIFICATION RESULTS

LR Model	Predicted Fault					
	Non-Faulty			Faulty		
	M1	M2	M3	M1	M2	M3
Non-Faulty	85	82	84	2	3	1
Faulty	2	11	1	58	49	59

TABLE VI. SENSITIVITY, SPECIFICITY AND ACCURACY

LR Model	Sensitivity (%)	Specificity (%)	Accuracy (%)
M_1	97	98	97
M_2	82	97	90
M_3	98	99	99

VII. MODEL DISCUSSIONS

In this section, the predicted model is assessed based the results presented in Table VI and VII. In Table VII presents the overall summary of the prediction model and a dash indicate the metric was not included in the model.

TABLE VII. MODEL PREDICTORS

Predicted Model	Predictors (λ)					Constant (ω)
	CBO	RFC	WMC	DIT	SLOC	
M_1	2.867	0.195	0.711	-	-	-36.938
M_2	-	-	0.173	-	0.008	-4.857
M_3	2.938	0.200	-	2.214	-	-37.124

In Table IV the maximum likelihood estimate of the model, M_3 which measures how poorly the model predicts FP of classes was very encouraging. The results indicates that the models is good in FP prediction since the smaller the statistic, the better the model. M_1 has a maximum likelihood value of 16.681, M_2 106.227 and M_3 17.605. Furthermore, the Cox & Snell R^2 values were very high though it is difficult to achieve a value of 1. It shows that the models are accurate in prediction since the higher the R^2 values, the higher the effect of the models. In addition, the R^2 values were all positive with $M_1 = 0.711$, $M_2 = 0.464$ and $M_3 = 0.709$. Based on the results, M_1 and M_3 have the highest R^2 values indicating the strength of the dependent variables in identifying the likelihood of a fault in OOS class.

Given the sensitivity, specificity and accuracy of the model, the values were also very high which indicates that, several non-faulty and faulty classes were appropriately classified. See Table VI. The high accuracy indicates the higher the quality of the software system under maintenance will be. This stems from the fact that detecting which class is faulty early, would allow mitigating plans to be concentrated on the high risk components before changes can be implemented.

VIII. MODEL APPLICATION TO SCIA

This section present the application of the prediction model during software maintenance. Using this model will assist in identifying which set of classes in the impact set of a change request are more likely to have faults or fail if changes are committed on them. To construct the prediction model, the KC1 dataset and the software metrics were used. However, before predicting FP of classes in the impact set, static analysis SCIA framework shown in Figure 1 will be applied to predict the impact set. Based on the variables collected, the maintainer can

then compute the FP of each class known to be affected by the change request using any of the prediction models M_1, M_2 and M_3. In this case, we employed M_3 as the predictive model since it yielded the highest accuracy rate of 99%. Thus, the model is fitted follows:

(1/Class FP) = (- 37.124 + 2.938(CBO) + 0.2(RFC) + 2.214(DIT))

Furthermore, a class is considered fault-prone if the risk is more than 50%. That is, the risk rate > cutoff value (0.5). To compute the risk associated with each class, we developed a novel tool called Class Change Recommender (CCRecommender).

A. Class Change Recommender

The CCRecommender is a novel tool developed in Java to help software maintenance personnel to compute the risk or FP of each class affected by a change request before the change implementation. To use CCRecommender during maintenance involve building the model first by using any statistical tool in order to compute values of the regression coefficients, ω, λ, and others parameters. Then the parameter values and the metric values of each included metric are use as input to the tool. In the M_3 model chosen, three metrics are included, CBO, RFC and DIT. The interface of the tool is captured in Figure 2. The prediction menu is where the actual computation is performed. First is the entering of the intercept (ω) called *alpha*. Next is the choice of the metrics for the model, the entering of their values as well as their respective regression coefficients (λ) called *beta*. After entering the value and the λ value for a metric, the button "SET" is used to input the values of the next chosen metric. After entering all the parameters values, the COMPUTE button is used to obtain the risk value in percentage of the affected class.

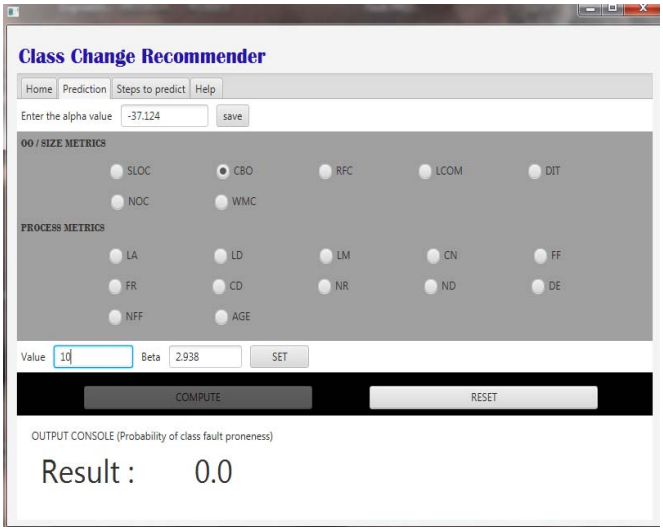

Fig. 2. CCRecommender system

In essence, CCRecommender also offers modification advice based on the risk value of each class on whether a change is advisable or not. It is modeled based on the following cutoff probabilities or conditions:

1) ***Condition I:*** If the risk value is less than or equals to 20, (prob. ≤ 0.2) a **GREEN CIRCLE** will appear. The indication is that the class is not fault-prone and the change can be implemented with no risk. For instance, the result captured in Figure 3 indicates that the risk associated with making changes to a class having CBO = RFC =10 and DIT = 1 is 3%.

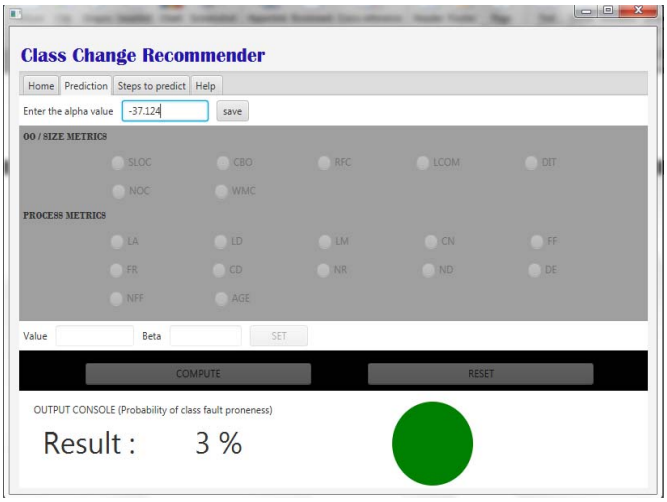

Fig. 3. Prediction results for condition I

2) ***Condition II:*** If the risk value is greater than 20% but less than or equals to 50% (0.2 > prob. ≤ 0.5), a **BLUE CIRCLE** will appear indicating the class is not fault-prone, but care must be taken when making changes to such a class. For instance, if a particular class has CBO = 10, RFC = 10 and DIT = 1, the prediction on the system will yield the following result as shown in Figure 4.

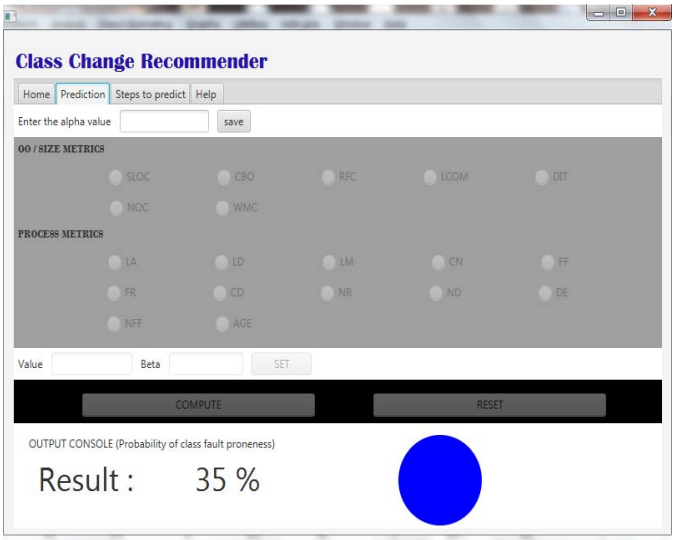

Fig. 4. Prediction results for condition II

3) ***Condition III:*** If the risk value is greater than 50% but less than or equals to 70%, (0.5 > prob. ≤ 0.7), a **YELLOW CIRCLE** will appear. The indication is that the class is fault-prone. However, before changes can be made verification and validation activities must be focused on such class to fix the faults. For example, the FP probability for a class having metric values, say CBO = 11, RFC = 25 and DIT = 0 will be 55% as shown in Figure 5.

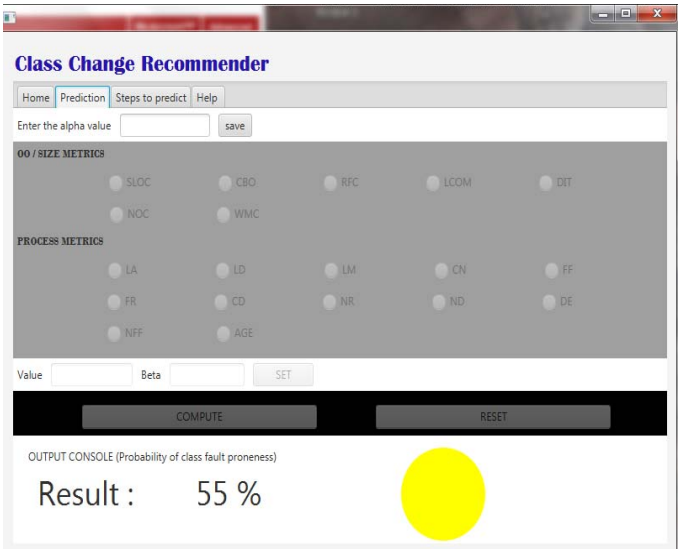

Fig. 5. Prediction results for condition III

4) ***Condition IV:*** lastly, if the risk value is greater than 70%, (prob. > 0.7), a **RED CIRCLE** would appear, indicating that the class is severely fault or failure-prone. In this case, change is not recommended on such as class. For instance, for a class having CBO = 8, RFC = 36 and DIT = 5, the risk rate or FP probability would be 99%. See Figure 6. However, additional plans can be invoked that either reject the change or redesign the system, apply refactoring and so on before actual changes can be possible.

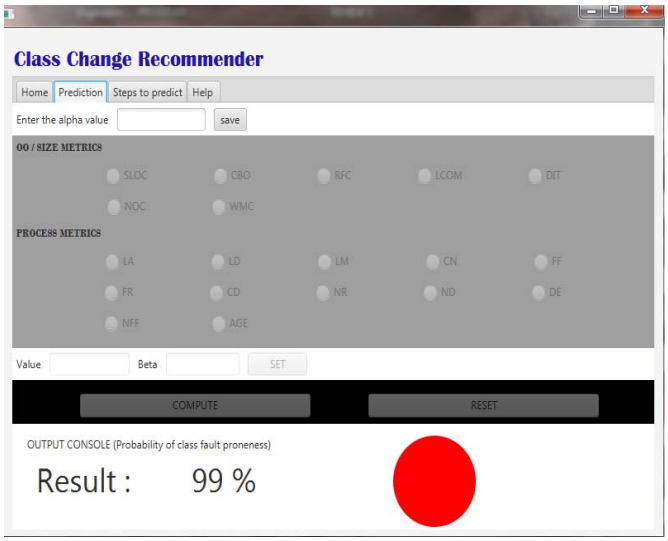

Fig. 6. Prediction results for condition IV

The above results indicates that, a class having a high amount of coupling, a great number of outside and inside methods coupled with inheritance having extended depth are more likely to have faults than a class with lower values of the metrics. Thus, we advise that software engineers should use the information provided by the model after SCIA in order to know if a change is going to be risky or not. This will assist in reducing or avoiding the costly changes.

IX. Conclusions and Future Work

This paper presented an extended framework for SCIA that incorporates both change impact and FP predictions during software maintenance. The approach is intended to preserve the quality of the system under maintenance. As today software applications have grown in size and complexity as well as OOS classes being faulty, maintaining a faulty class without the knowledge of existing fault could be very risky. As a cost-effective, we discussed an approach that predict the occurrence of potential faults in the impact set before actual changes are made. To assist engineers in channeling efforts to high risks components during maintenance, a tool called CCRecommender was developed. It is intended for use to quantify the risk associated with making changes on a fault-prone class in the impact set before actual changes are implemented. This will provide important information in making good decisions, plan and allocate resource, reduce cost as well as assure high quality.

The study limitation is that CCRecommender has not been applied to a real-world system. We only used the dataset from the public domain to demonstrate its operation during maintenance. Therefore, the results obtained cannot be generalized to any other similar empirical studies in terms of metrics validation. Moreover, the tool is not automated, thus require parameters to be mined externally. To this end, the future work will be on automation and to perform the prediction on a real-world software in order to evaluate the accuracy and effectiveness of the tool. Additionally, we will consider other prediction models to find out which one is more effective and accurate.

Acknowledgment

The study reported in this paper has been conducted with the assistance from MaSIM in the NWU. We are very appreciative and grateful for the opportunity given to us and we would like to thank their efforts in making this study a huge success.

References

[1] Xu, J., Ho, D. and Capretz, L.F. An Empirical Validation of Object-Oriented Design Metrics for Fault Prediction. Journal of Computer Science No.4, Vol 7, pp. 571-577, 2008. ISSN 1549-3636

[2] Subramanyam, R. and Krishnan, M.S.: Empirical Analysis of CK Metrics for Object- Oriented Design Complexity: Implications for Software Defects. IEEE Trans. Software Eng. No.29, pp. 297-310, 2003

[3] Zhou, Y., & Leung, H. Empirical analysis of object oriented design metrics for predicting high and low severity faults. IEEE Transactions on Software Engineering, 32(10), pp. 771–784, 2006

[4] Singh, Y. Kaur, A. and Malhotra, R. Empirical validation of object-oriented metrics for predicting FP models. Software Quality Journal, vol.18 pp. 3–35, 2010

[5] Succi, G., Pedrycz, W., Stefanovic, M., Miller, J.: Practical assessment of the models for identification of defect-prone classes in object-oriented commercial systems using design metrics. Journal of Systems and Software 65, pp. 1-12, 2003

[6] Myers, G., Badgett, T., Thomas, T., Sandler, C.The Art of Software Testing, second ed. John Wiley & Sons, Inc., Hoboken, NJ., 2004.

[7] Fenton, N., Ohlsson, N. Quantitative analysis of faults and failures in a complex software system. IEEE Transactions on Software Engineering, Vol. 26 no. 8, pp.797-814, 2000

[8] Shatnawi, R. andLi, W. The effectiveness of software metrics in identifying error-prone classes in post-release software evolution process. The Journal of Systems and Software vol. 81 pp.1868–1882, 2008.

[9] Jönsson, P. and Lindvall, M.: "Impact Analysis" Engineering and Managing Software Requirements Issue: 6, Springer-Verlag, pp. 117-142, 2005

[10] Bohner, S. A.: "Extending software change impact analysis into COTS components" Proceedings of the 27th Annual NASA Goddard Software Engineering Workshop, Greenbelt, USA, pp.175 -182, 2000

[11] Emam, K.E., Melo, W.L., Machado, J.C.: The prediction of faulty classes using object-oriented design metrics. Journal of Systems and Software No. 56, pp. 63-75, 2001

[12] Malhotra, R., Kaur, A. and Singh, Y. Empirical validation of object-oriented metrics for predicting FP at different severity levels using support vector machines. International Journal System Assurrance Engineering Management. No.1, vol. 3, pp. 269–281, 2010.

[13] Isong, B.E. and Ekabua, O.O. (2013) "A Systematic Review of the Empirical Validation of Object-oriented Metrics towards Fault-proneness Prediction". International Journal of Software Engineering and Knowledge Engineering (IJSEKE) WSPC. Vol. 23, No. 10. pp. 1513–1540 DOI: 10.1142/S0218194013500484. ISSN: 0218-1940

[14] Metrics Data Program (MDP, 2006), http://sarpresults.ivv.nasa.gov/ViewResearch/107.jsp

[15] Ruchika Malhotra, Nakul Pritam Yogesh Singh On the Applicability of Evolutionary Computation for Software Defect Prediction. International Conference on Advances in Computing, Communications and Informatics (ICACCI, 2014). Pp. 2249 – 2257, 2014

[16] Wu, Y., Zhao, Y. and Lu, H. The Influence of Developer Quality on Software Fault-Proneness Prediction, 2014 Eighth International Conference on Software Security and Reliability (SERE),. pp.11-19, 2014

[17] Chen, J. and Huang, S. An empirical analysis of the impact of software development problem factors on software maintainability. The Journal of Systems and Software 82 (2009) 981–992

[18] Holgeid, K.K., Krogstie, J., Sjøberg, D.I.K. A study of development and maintenance in Norway: assessing the efficiency of information systems support using functional maintenance. Information and Software Technology 42 (10), 687–700, 2000.

[19] Sun, X., Li, B., Tao, C., Wen, W. and Zhang, S. "Change Impact Analysis Based on a Taxonomy of Change Types" 2010 IEEE Proceedings of 34th Annual Computer Software and Applications Conference (COMPSAC'10), pp.373-82, 2010.

[20] Chidamber, S., Kemerer, C.F.(1994): A metrics suite for object oriented design. IEEE Trans. Softw. Eng. Vol. 20, No. 6, pp. 476–493, 1994.

[21] Yu, P., Systa, T., & Muller, H. Predicting FP using OO metrics: An industrial case study. In Proceedings of Sixth European Conference on Software Maintenance and Reengineering, Budapest, Hungary, pp.99–107, 2002

[22] Arisholm, E. Briand, L.C. and Johannessen, E.B. "A systematic and comprehensive investigation of methods to build and evaluate fault prediction models", The Journal of Systems and Software, vol.83, pp.2–17, 2010

[23] Gyimóthy, T., Ferenc, R., Siket, I.: "Empirical Validation of Object-Oriented Metrics on Open Source Software for Fault Prediction". IEEE Transactions on Software Engineering, No.31, pp.897-910, 2005

[24] Aggarwal, K. K., Singh, Y., Kaur, A. and Malhotra, R. Empirical Analysis for Investigating the Effect of Object-Oriented Metrics on FP: A Replicated Case Study. Software Process Improvement and Practice, No.14, pp. 39–62, 2009.