I.J. Information Technology and Computer Science, **2014, 02, 1-13**
Published Online January 2014 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijitcs.2014.02.01

A Machine Learning based Efficient Software Reusability Prediction Model for Java Based Object Oriented Software

Surbhi Maggo
Jaypee Institute of Information Technology, Noida, India
E-mail: surbhi.maggo@gmail.com

Chetna Gupta
Jaypee Institute of Information Technology, Noida, India
E-mail: chetnagupta04@gmail.com

***Abstract*—** Software reuse refers to the development of new software systems with the likelihood of completely or partially using existing components or resources with or without modification. Reusability is the measure of the ease with which previously acquired concepts and objects can be used in new contexts. It is a promising strategy for improvements in software quality, productivity and maintainability as it provides for cost effective, reliable (with the consideration that prior testing and use has eliminated bugs) and accelerated (reduced time to market) development of the software products. In this paper we present an efficient automation model for the identification and evaluation of reusable software components to measure the reusability levels (high, medium or low) of procedure oriented Java based (object oriented) software systems. The presented model uses a metric framework for the functional analysis of the Object oriented software components that target essential attributes of reusability analysis also taking into consideration Maintainability Index to account for partial reuse. Further machine learning algorithm LMNN is explored to establish relationships between the functional attributes. The model works at functional level rather than at structural level. The system is implemented as a tool in Java and the performance of the automation tool developed is recorded using criteria like precision, recall, accuracy and error rate. The results gathered indicate that the model can be effectively used as an efficient, accurate, fast and economic model for the identification of procedure based reusable components from the existing inventory of software resources.

***Index Terms*—** LMNN; Machine Learning; Metric; Procedure Oriented; Reusability

I. Introduction

The role of computer software has changed significantly over past 50 years. A society that is highly dependent on software and increasingly intolerant of software failures has placed immense pressure on software professionals for the development of more and more sophisticated and complex software systems. The growth in the expected sizes of software systems required is exponential, even with the rise in the number of computer professionals and the rise in technology that provides for improved hardware and computing performance and vast increases in memory and storage capacity, it has become difficult to keep the rise in software productivity in pace with the high demands for even more complex systems and maintenance of the existing software. Several decades of intensive research in fields of software engineering left software reuse as the only realistic and technically feasible approach to bring about the desired improvements in quality and productivity required by the software industry.

Software reuse, although simple in concept (creation of new software systems using the existing software artifacts), it offers a great deal of potential in terms of software productivity and software quality [1]. The formal idea of software reuse instituted the development of industry of reusable software components and the industrialization of the production of application software from off-the-shelf components, as proposed by Mcllory in [2]. Not only is reuse a promising strategy for increasing quality and productivity in the software industry, but a good software reuse process also provides for increased reliability and dependability, reduced process risk, decreased cost of implementation and time to market (accelerated development), effective use of specialist and standard compliance [3]. Reusability also increases the likelihood that prior testing and use has eliminated bugs thus delivering error free software codes. Thus it can be stated that software reusability is a measure of the ease with which

previously acquired concepts and objects can be used in new contexts [4]. The measure of this ease of reuse depends on certain attributes that make a software artifact/asset less or more reusable. Certain attributes that increase the likelihood of reusability of a component include high cohesion, loose coupling, modularity, ease of understanding and modification, separation of concerns and information hiding, proper documentation and low complexity [5].

For a long time since the introduction of the concept of reuse, source code was considered the only software artifact that can be reused. Recently other software products like algorithms, estimation templates, requirements, plan and design, documentation, human interface, user manuals and test suites etc are being considered for reuse [6]. Software reusability has been an active area of research in software engineering over the past 35 years but challenges and open questions still exists. Even with all the perceived benefits there are a number of risks and challenges associated with reusability like increased maintenance costs, lack of tool support, not-invented-here-syndrome [4], creating and maintaining a component library and finally finding, understanding and adapting reusable components. Reusing a non reusable component may cause severe loss in terms of time, cost and effort. Many researchers have worked to reduce the risks associated with reuse and make it faster, easier, more systematic, and an integral part of the normal process of programming. At the time of conception of reuse and its application in software industry researchers came up with a number of empirical methods of measuring reusability of components based on different versions of complexity and other metrics as mentioned in related work. Although the focus of current research in the field of software reuse is to establish meaningful relationships between these metrics (reusability attributes) in order to generate reliable functions that can efficiently evaluate the reusability levels and identify reusable components. Domains like data mining, machine learning, neural networks etc. are very useful in generating such functions. A tabular representation of many such proposed metrics and measures from literature is presented in section 2.

The presented work aims to systematize and ease software reusability process and reduce the risks associated with it by providing for an efficient automation model for software reusability prediction. The presented approach works for the identification and evaluation of software components to measure the software reusability levels (low, medium and high) of function based object oriented software systems. It uses a metric framework for the functional analysis of the object oriented software components that target essential attributes for reusability prediction also taking into account Maintainability Index (MI) to account for partial reuse as well. Further machine learning is explored to establish relationships between the functional attributes. The model works at functional level rather than at structural level. The performance of the automation tool developed namely, Java based Reusability Prediction Model using LMNN (JRPML), is recorded using criteria like precision, recall, accuracy and error rate. The results gathered indicate that the model can be effectively used as an efficient, accurate, fast and economic model for the identification of procedure based reusable components from the existing inventory of software resources.

The remaining paper is organized as follows: In the next section, related work is presented. Problem formulation and proposed automation model, methodology is described in section 3 and 4 respectively. In section 5 we present some experimental results of proposed JRPML tool implemented in Java. Section 6 presents comparison results of JRPML with other existing approaches including metrics used, technique used results of precision, recall, accuracy and classification error values etc. Section 7 discusses the application of proposed approach and finally section 8 presents the conclusion.

II. Related Work

Over the years, various attempts have been made in the field of measuring reusability to help developers identify reusable software components. The initial era of research, provided for a number of empirical approaches in the form of metrics and metric suites for reusability measurement. Although this trend has changed over last 5-7 years and concepts like data mining, machine learning, neural networks etc have come into picture, that have helped in better and more efficient identification of reusable components by generating functions that establish meaningful relationships among various reusability attributes. These domains use the earlier proposed metrics as the basis of reusability prediction and further generate functions wherein these metrics collectively work towards the determination of reusability. A summary/list of the earlier proposed empirical models approaches and recently researched upon intelligent approaches has been presented in Table 1 above.

The metrics presented in Table 1 use objective and quantifiable software attributes as their basis. Prieto-Diaz and Freeman [7] proposed 4 module-oriented metrics: program size, structure, documentation and language and a fifth metric named reuse experience to modify the first four. Selby [8] in his study identified a number of reusability characteristics like simple interface, less input – output etc, using data from a NASA software environment. Chen and Lee [9] developed 130 reusable C++ components and performed a controlled experiment using these to relate quality and level of reuse. Caldiera and Basili [10] provided for 4 metrics namely Cyclomatic complexity, Halstead's program volume, Regularity metric and Reuse frequency that help in quantifying reusability

attributes. The ESPRIT-2 project [11] REBOOT (Reuse Based on Object-Oriented Techniques) developed a taxonomy of reusability attributes. It listed four reusability factors, a list of criteria for each factor, and a set of metrics for each criterion. Three approaches; function, form and similarity were presented by Hislop [12] for evaluating software.

Table 1: Summary of proposed Approaches

Empirical Metric Based Approaches	Recent Intelligence Based Approaches		
	Metric Used	Algorithm Used	Proposed By
Prieto-Diaz and Freeman[7]	CB[10]	Neural Network	Manhas et al. [21]
Selby[8]	CK[17]	Hybrid k-Means & Decision Tree	Shri et al. [22]
Chen and Lee[9]	CB	SVM	Kumar [23]
Caldiera and Basili[10]	CB	k-NN	Cheema et al. [24]
REBOOT[11]	CB	DBSCAN	Saini et al. [25]
Hislop[12]	S/W Metric[18]	Hierarchical	Czibula et al. [18]
Boetticher and Eichmann[13]	CK	Fuzzy Neuro	Sandhu et al. [26]
Torres and Samadzadeh[14]	CK	k-Means	Sandhu et al. [27]
Mayobre[15]	S/W Metric[19]	Expectation Maximization	Goel et al. [19]
US Army Reuse Centre [16]	S/W Metric[20]	k-Means	Kanellopoulos et al.[20]

Boetticher and Eichmann [13] trained a neural network to mimic set of human evaluators and generated 250 code parameters for reusability assessment. Torres and Samadzadeh [14] examined effects of information theory metrics: entropy loading and control structure entropy on software reusability. Mayobre [15] described Code Reusability Analysis (CRA) for the identification of reusable work products in existing code. CRA uses three methods: Caldiera and Basili [15], Domain Experience Based Component Identification Process (DEBCIP) and Variant Analysis Based Component Identification Process (VABCIP) for reusability assessment. The Army Reuse Center (ARC) inspects all software submitted to the Defense Software Repository system (DSRS) [16].

The reusability evaluations [16] under the inspection include 31 metrics in the initial stage and around 150 in the final stage. In recent years a number of intelligent system models have been proposed for reusability evaluation. Manhas et al. [21] used functional metrics proposed by Caldiera and Basili (CB) along with a number of back propagation based neural network algorithm and provided for their comparative results. A similar Neuro fuzzy approach was used by Sandhu et al. [26] on structural Chidamber and Kemerer (CK) metric suite. Sandhu et al. [27] also proposed the use if basic clustering algorithm k-Means for faulty module prediction again using the structural CK metric suite as the basis. A hybrid of the basic k-Means along with decision tree for reusability evaluation was presented by Shri et al. [22] in their research. The presented model was more efficient in comparison to the basic k-Means algorithm. A metric framework along with k-Means was employed by Kanellopoulos et al. [20] for evaluating the maintainability and hence reusability of OO software systems. The proposed solution was semi automated as the parsing engine extracted the data from the source code and stored them on a database. Caldiera and Basili metric suite provided for the functional analysis in numerous works. Cheema et al [24] and Kumar et al. [23] used the suite along with k-NN clustering algorithm and support vector machine (SVM) respectively in their proposed models. Appreciable accuracy rates were achieved by Cheema et al. [24], while those with SVM depended upon the training set. Saini et al. [25] also used the metric suite along with DBSCAN (Density Based Spatial Clustering of Applications with Noise). It highlighted the concept of density of components as important for reuse evaluation. Although few of the proposed approaches have presented satisfactory results but none of them presents a reliable and completely automated solution for the evaluation and identification of reusable components as is done by the JRML model presented.

III. Problem Formulation

Software reuse not only improves productivity but also enhances the quality, reliability and maintainability of the software products/components. Reuse acts as a major boon for software development organizations as it makes the software development process cost and time efficient, while helping the organizations deliver almost error free codes to its clients without much effort, as the code is already tested many times during its earlier reuse [24] . For the organizations, where the concept of software reuse has not been realized yet, there are two possible approaches towards software reuse: a) to develop codes from scratch that can be reused at a later stage, b) to identify and further extract reusable code snippets/components from the already existing inventory of resources. But there exists an additional cost required for the development of reusable software components from scratch that can be used to build and strengthen their software reservoirs. This extra cost can be avoided by employing the second approach towards software reusability, of identifying reusable components from existing resources. A number of metrics and measures have been proposed in

literature for evaluating the reusability levels of a component as discussed in section (related work). But the issue of how these metrics and measures collectively determine the reusability levels is relatively less explored and is still in need of an efficient and reliable automated solution.

In this paper we propose an efficient prediction model for the identification of reusable software components that is based upon software metrics that target function oriented components and a statistical machine learning based clustering approach (LMNN: Large margin nearest neighbor) to map relationships among the reusability metrics and attributes. The learning process followed by the model makes the prediction far more accurate and efficient, providing for promising results in predicting reusability levels. Software engineers can thus use this Java based model for efficient identification of procedure based reusable components which works at functional level rather than at structural level for predicting efficient and precise reusability levels of function oriented object oriented software.

IV. Solution Framework

The proposed solution framework for the reusability evaluation model is presented in Fig. 1 below. The solution framework presented in the paper constitutes of three basic modules as follows:

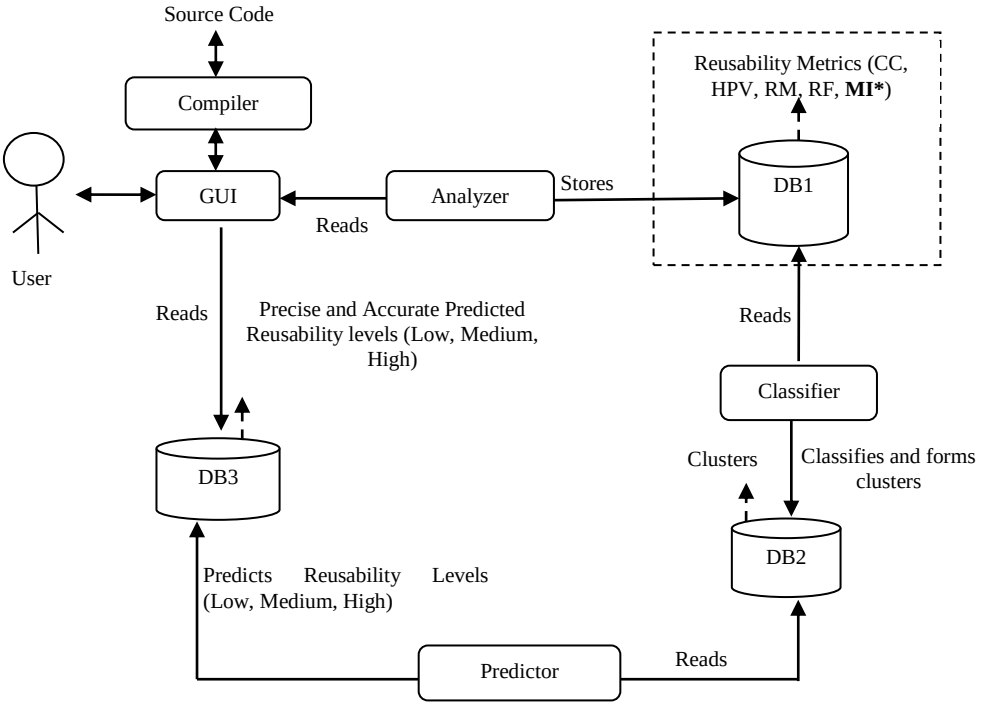

Fig. 1: Solution Framework

Module1: Analyzer - The process starts with Metric selection, which provides the analyzer module with the appropriate metrics that are to be calculated for the reusability prediction model. The reusability prediction model presented in the paper attempts to select metrics that characterize reusability attributes either by directly providing a measure for them or indirectly through measures of evidence of an attribute's existence. These reusability attributes that are responsible for making a component reusable in another system is its quality, low reuse cost and its significant functional usefulness in the context of the application domain. The prediction model is based on the automation of the selected metrics in order to generate values of these metrics for the software components under test. A set of acceptable values is defined for each of the metrics. These values can be either simple ranges of values (measure is acceptable between al and a) or more sophisticated relationships among different metrics.

The five metrics selected for our approach are, Cyclomatic complexity, Halstead Software Science Indicator, Regularity Metric, Reuse Frequency and Maintainability Index. All these metrics produce values that are interpreted by various researchers in different ways for predicting reusability values [10, 28].

Table 2 below presents the reusability attributes namely, Usefulness, quality and cost along with the factors that influence these attributes. In the table below, with all the factors under each reusability attribute, there are associated metric measures. These metrics directly measure the factor or indirectly predicts the likelihood of its presence at functional level. In this paper we have taken maintainability index to gather precise values for reusability.

Table 2: Factors influencing Reusability

Reusability Attribute	Metric Measures
Usefulness	
Variety of Functions Commonality of Functions -Within a System -Within a Domain -Overall	Cyclomatic Complexity, Regularity Metric Reuse Frequency Reuse Frequency
Quality	
Ease of Modification Correctness Testability Readability Performance -Time -Space	Maintainability Index Halstead Volume Cyclomatic Complexity Maintainability Index Maintainability Index
Cost	
Packaging Use in New System -Retrieval -Integration -Modification Extraction - Identification - Qualification	Cyclomatic Complexity, Halstead Volume Regularity Metric Halstead Volume, Regularity Metric, Maintainability Index Halstead Volume Halstead Volume

Here in our presented reusability evaluation model we investigate and use maintainability index as our fifth metric to account for maintenance efforts in case of partial reuse. MI is an effective way of assessing software thereby identifying and quantifying software's maintainability as it provides an excellent guide to direct human investigation and identifies components with designs closer to problem domain and greater reliability and readability thus having a positive impact on reusability of such software components [28]. Using MI in the presented system helps in the identification of reusable software components that are not only applicable for direct reuse but can also be reused partially or after modifications with ease and in a very cost and time effective way.

The five selected metrics for the reusability evaluation model are described in detail in the following section.

1) Cyclomatic Complexity – It is software metric that indicates the complexity of a program using its control flow graph. According to Mc Cabes [29], the value of Cyclomatic Complexity (CC) can be obtained using the following equation:

$$CC = Number\ of\ Predicate\ Nodes + 1 \quad (1)$$

Number of predicate nodes in the equation 1 refer to the decision nodes in the component code such as if-else, for, while statements

2) Halstead Software Science Indicator – According to this metric volume [30] of the source code of the software component is expressed in the following equation:

$$Halstead\ Volume = N1+N2\log_2(\eta 1+ \eta 2) \quad (2)$$

where, η1 is the number of distinct operators that appear in the program, η2 is number of distinct operands that appear in the program, N1 is the total number of operator occurrences and N2 is the total number of operand occurrences.

3) Regularity Metric [10] - The notion behind Regularity is to predict length based on some regularity assumptions. Regularity is the ratio of estimated length to the actual length. As actual length (N) is sum of N1 and N2. The estimated length is shown in the following equation:

$$Estimated\ Length = N' = \eta 1\log_2 \eta 1 + \eta 2\log_2 \eta 2 \quad (3)$$

The closeness of the estimate is a measure of the Regularity of Component coding is calculated as:

$$Regularity = 1 - \{(N-N')/N\} = N'/N \quad (4)$$

4) Reuse Frequency – Reuse frequency is calculated by comparing number of static calls addressed to a component with number of calls addressed to the component whose reusability is to be measured.

$$\text{Reuse-frequency} = \frac{\eta(C)}{\frac{1}{M}\sum_{i=0}^{M}\eta(S_i)} \quad (5)$$

5) Maintainability Index – Maintainability Index is a software metric which measures how maintainable (easy to support and change) the source code is. The maintainability index is calculated as a factored formula:

$$MI = 171-5.2*\ln(V)-0.23*(G)-16.2*\ln(LOC) \quad (6)$$

where MI refers to the maintainability index and *ln* refers to natural logarithm function. LOC is the lines of Codes, G is Cyclomatic complexity and V is volume of code.

The Analyzer modules takes input from the GUI and generates the values of the metrics for the five selected metrics for all the software components taken from the input system provided. For the first two metrics i.e. Cyclomatic Complexity and Halstead Program Volume, the values generated do not have any specified range, as they directly depend on the component size. Hence for the analysis of these metrics required for the further processing of the system model, it is required to normalize these values to bring them to a particular range. Here we have normalized them to a range of 0-10. The values for other three metrics - regularity metric,

reuse frequency and maintainability index, already lie in a pre defined range i.e. 0-1 for regularity and reuse frequency and 0-100 for maintainability index.

Module 2: Classifier - We use a statistical machine learning [31] algorithm named Large Margin Nearest Neighbor (LMNN)[33] to classify the components into different reusability levels (low, medium or high) on the basis of the values of the metrics generated for the components under the previous module. The algorithm is a variant of k-NN i.e. k Nearest Neighbors[33] that provides for better accuracy as compared to the basic k-NN. The accuracy of the k-NN classification depend significantly two factors: 1) On the metric used to compute the distances between two points (here two software components, with distances being the differences in their respective reusability metrics), 2) On the population of the examples of each class present in the sample space. LMNN is based on pseudo learning designed for k-Nearest neighbor classification. This pseudo learning works on the factors affecting k-NN's accuracy using the Mahalanobis metric. The Mahalanobis metric [34] can be viewed as a global linear transformation of the input space that precedes k-NN classification using Euclidean distances. In this approach, the metric is trained with the goal that the k - nearest neighbors always belong to the same class while examples from different classes are separated by a large margin. Fig. 2 from [32] presents a schematic illustration of LMNN. The Mahalanobis distance metric is obtained as the solution to a semi-definite program [35]. On several data sets of varying size and difficulty, the metrics trained in this way lead to significant improvements in k-NN classification. Sometimes these results can be further improved by clustering the training examples and learning an individual metric within each cluster.

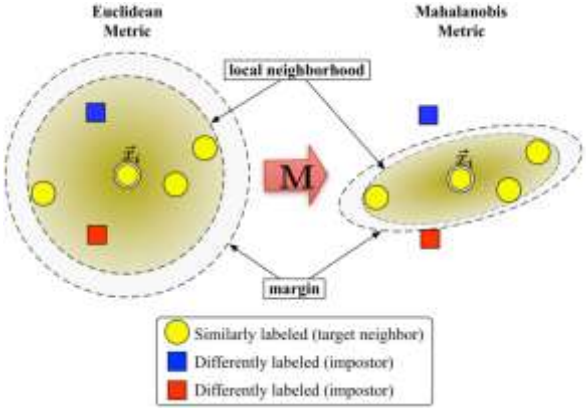

Fig. 2: LMNN Schematic Illustration [33]

LMNN improves the basic Euclidian Metric and adjusts the distances of a point with its neighbors depending on the similarity or dissimilarity of the class labels, thus improving the efficiency and accuracy of the reusability prediction results, reducing the risks associated with incorrect reusability predictions.

Literature presents a number of metrics [36] for reusability prediction and evaluation that reckon different reusability attributes, but for more precise and exact evaluation of reusability we need to take all reusability attributes into consideration. Hence it is highly required to establish functions that provide for meaningful relationships among these attributes for reusability prediction. The presented classification algorithm helps us develop these relationships, and with its pseudo learning based training approach it makes the analysis and relation establishment easier. It not only makes the evaluation process easy and fast but also very efficient.

Module 3: Reusability Predictor – The output reusability levels for the software components identified are evaluated on the basis of the classification done using LMNN in the module 2. Now we use confusion matrix [37] in order to assess the results generated by the statistical machine learning algorithm implemented. Confusion matrix (summarized in Table 3 below) is an important tool for evaluating the prediction results as it presents the results in an easy to understand way and make it account for the effects of wrong predictions. To evaluate the efficiency and accuracy of our presented model we have generated the confusion matrix and following four parameters have been calculated for the generated confusion matrix:

Table 3: Reusability Prediction [37]

Parameter	***Remarks***	***Formula***
Precision	It is the fraction of components identified by the system that are relevant.	*Precision = True Positive/ (True Positive + False Positive)*
Recall	It is the fraction of relevant components that the system could identify.	*Recall = True Positive/ (True Positive + False Negative)*
Accuracy	The percentage of the predicted values that match with the expected values of the reusability for the given data.	*Accuracy = (Correctly Classified/ Total Classified)*100*
Classification Error	The percentage of the predicted values that does not match with the expected values of the reusability for the given data.	*Classification Error = (Incorrectly Classified/ Total Classified)*100*

High accuracy, high precision, high recall and low classification error value represents the best system.

V. Result Observation and Discussion

Following steps were followed for collection of relevant data in this study. The implementation of the tool is done in Java using NetBeans IDE 7.0.1. For the proposed model (reusability evaluation tool) data

collection is done for over 175 object oriented code fragments. The tool provides GUI based interface for interaction with the user. User is required to provide the codes as an object oriented system to the tool JRPML, from which the different components are identified for reusability level prediction. The evaluation process includes meta data generation using metrics presented in the Analyzer Module in Section 4. The generated dataset is further analyzed using a pseudo learning based statistical machine learning algorithm LMNN and reusability classes are predicted for all the identified components. Computed results are evaluated in terms of precision, recall, accuracy and classification error. Following section discusses the results of step by step data collection and processing at functional level using JRPML followed by a comparison of results obtained by our technique with already existing techniques.

Step 1: The tool requires the user to provide an object oriented system as input. The individual components (classes or individual code segments) are identified and extracted from the given set of input by the tool. The same is presented in Fig. 3 below.

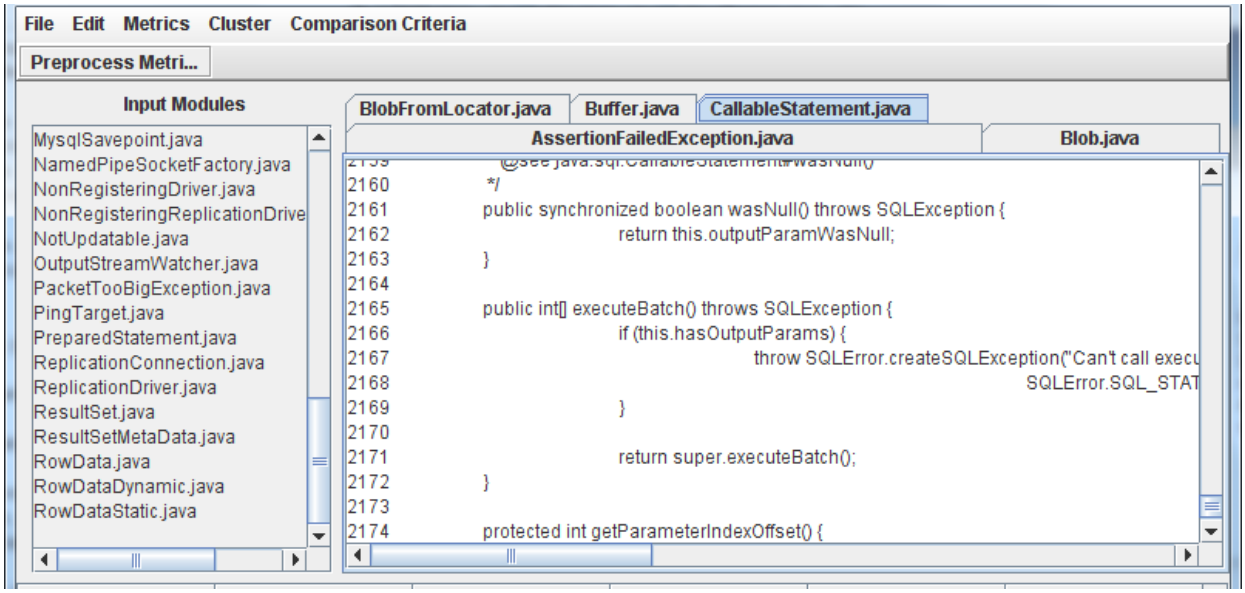

Fig 3: Snapshot view of user interface

Step 2: Next step is to generate meta-data for the five selected metrics (discussed in analyzer module of section 4) corresponding to all components identified. Meta data view generated for each metric as dataset shown in Fig. 4.

Component	Cyclomatic Complexity	Halstead Software Sci...	Regularity Metric	Reuse-Frequency Met...	Maintainability Index
40	6	388.0	0.7835	0.7657	77
41	6	156.0	0.5	0.4822	89
42	982	210552.0	0.1301	0.1123	15
43	97	11658.0	0.3201	0.3023	1
44	18	1155.0	0.5758	0.558	61
45	2633	559539.0	0.0804	0.0626	15
46	329	23303.0	0.2403	0.2225	64
47	58	3355.0	0.3398	0.322	29
48	93	7008.0	0.4195	0.4017	6

Fig. 4: Snapshot of Meta Data view of Metrics as dataset

Step 3: Next step is to normalize meta-data obtained from step 2. It can be seen clearly in Fig. 4 that the values of metric 3, 4 and 5 are in the range of 0-1 and 0-100 respectively. The values of metric 1 and 2 donot have a predefined range as the values of Cyclomatic complexity and Halstead Volume vary according to the size of the component. Hence we normalize these values and set them in a range of 0-10. The normalized view is presented in Fig. 5.

Step 4: In the next step the values of metrics from step 3 are raised in a hierarchical level of interpretation to categorical text using [10] and [38]. The numeric metric values are interpreted as low, medium or high using [10] and [38].This interpretation is shown in Fig. 6. The reusability class labels are defined as high, low or medium based on the categorical metric data of the respective components.

Component	Cyclomatic Complexity	Halstead Software Sci...	Regularity Metric	Reuse-Frequency Met...	Maintainability Index
40	1.0034	1.0037	0.7835	0.7657	77
41	1.0034	1	0.5	0.4822	89
42	4.3459	4.3851	0.1301	0.1123	15
43	1.3151	1.1851	0.3201	0.3023	1
44	1.0445	1.0161	0.5758	0.558	61
45	10	10	0.0804	0.0626	15
46	2.1096	1.3724	0.2403	0.2225	64
47	1.1815	1.0515	0.3398	0.322	29
48	1.3014	1.1102	0.4195	0.4017	6

Fig. 5: Snapshot of normalized meta-data obtained from step 2

Component	Cyclomatic Comple...	Program Volume	Regularity	Reuse-Frequency	Maintainability
16	High	High	Low	Low	Medium
17	Medium	Medium	Low	Low	High
18	Low	Low	Medium	High	High
19	Low	Low	Medium	High	High
20	Medium	Low	Low	Low	Medium
21	Low	Low	Medium	High	High
22	Low	Low	Medium	Medium	High
23	Low	Low	High	High	High
24	Medium	Medium	Medium	Medium	High
25	Low	Low	High	High	High
26	Medium	Low	Medium	Medium	Low

Fig. 6: Snapshot Categorical view of Meta data

Step 5: In order to establish meaningful relationships between software metrics calculated in the previous steps for reusability prediction our tool uses a statistical machine learning approach [31] on the generated metadata. The goal is to create a model that predicts the reusability values of a target variable (component) based on several input variables (metrics). We use LMNN [32] algorithm in our model. For the implementation of the algorithm the user is asked to input percentage for train and test data as shown in Fig. 7. The training set is classified using three class labels namely, high, medium and low level of reusability depending upon the combination of metric values for the components. The snapshot of the extended reusability dataset along with the generated class labels is illustrated in Fig. 8. The model is trained using the dataset presented in Fig. 8 and Mahalanobis metric [34]. Fig. 9 presents the test data for which the reusability levels are to be predicted using LMNN. Fig. 10 shows the snapshot of JRPML tool for test data result with actual and predicted classes.

Fig. 7: Snapshot view of selecting training data by user

Component	Cyclomatic Comple...	Program Volume	Regularity	Reuse-Frequency	Maintainability	Reusability
16	High	High	Low	Low	Medium	Low
17	Medium	Medium	Low	Low	High	Low
18	Low	Low	Medium	High	High	High
19	Low	Low	Medium	High	High	High
20	Medium	Low	Low	Low	Medium	Medium
21	Low	Low	Medium	High	High	High
22	Low	Low	Medium	Medium	High	High
23	Low	Low	High	High	High	High
24	Medium	Medium	Medium	Medium	High	Medium
25	Low	Low	High	High	High	High
26	Medium	Low	Medium	Medium	Low	High

Fig. 8: Snapshot of extended reusability dataset with reusability classes

Component	Cyclomatic Com...	Program Volume	Regularity	Reuse-Frequency	Maintainability
12	High	High	Medium	Medium	Medium
13	Low	Low	High	High	High
14	Low	Low	Medium	High	High
15	Medium	Low	Medium	Medium	Low
16	High	High	Low	Low	Medium
17	Medium	Medium	Low	Low	High
18	Low	Low	Medium	High	High
19	Low	Low	Medium	High	High
20	Medium	Low	Low	Low	Medium
21	Low	Low	Medium	High	High

Fig. 9: Snapshot of Test Data for prediction

Component	Cyclomatic Com...	Program Volume	Regularity	Reuse-Frequency	Maintainability	Actual Class	Predicted Class
12	High	High	Medium	Medium	Medium	Low	Low
13	Low	Low	High	High	High	High	High
14	Low	Low	Medium	High	High	High	High
15	Medium	Low	Medium	Medium	Low	High	High
16	High	High	Low	Low	Medium	Low	Low
17	Medium	Medium	Low	Low	High	Low	Medium
18	Low	Low	Medium	High	High	High	High
19	Low	Low	Medium	High	High	High	High
20	Medium	Low	Low	Low	Medium	Medium	Medium
21	Low	Low	Medium	High	High	High	High

Fig. 10: Snapshot of Test data result with actual and predicted classes

Step 6: For assessing the performance of the presented approach the results generated for the test data are evaluated using the confusion matrix. The confusion matrix corresponding to the machine learning functions used is presented in Table 4(a). Based on this matrix the prediction model is assessed by generating its accuracy and classification error values along with precision and recall values for each reusability class. Table 4(b) presents the values for these evaluation parameters. The accuracy and classification error values obtained using our approach is 94.2% and 5.8% respectively. Fig. 11 shows number of components (used in this study, details are provided in section 4) classified in three categories namely low reusability, medium reusability and high reusability. With the obtained values of accuracy (around 94%) and classification error (around 6%) the tool JRPML seems to possess great potential of detecting the components according to reusability levels with a very high probability of the levels being correct. This helps greatly in reducing the probability of false alarms minimizing the risks associated with identifying a negligibly reusable component as reusable thus preventing the wastage of time, effort and cost required in its reuse. Hence the developed tool is a safe, reliable and effective way of assessing and identifying reusable components from existing reservoirs.

Table 4(a): Predicted Reusability Classes using confusion matrix [37]

		Predicted Reusability Classes		
Actual Reusability Classes		Low	Medium	High
	Low	16	1	0
	Medium	0	10	2
	High	0	1	40

Table 4(b): Results of precision, recall, accuracy, classification error using JRPML

Reusability Classes	Precision	Recall	Accuracy	Classification Error
Low	1	.94	94.2%	5.8%
Medium	0.83	.83		
High	0.95	.98		

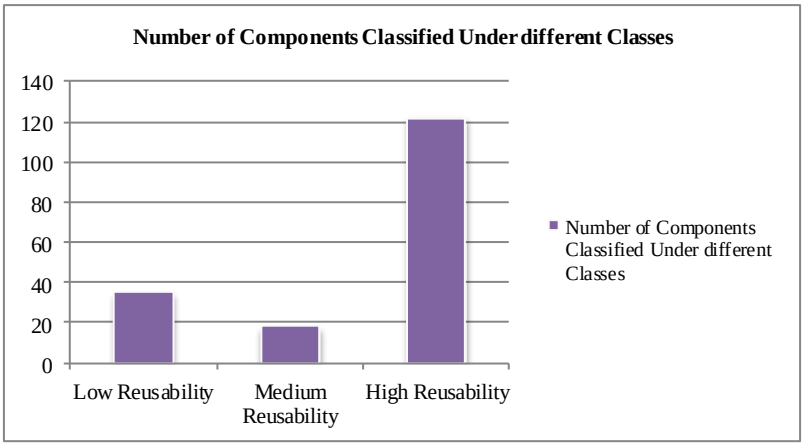

Fig. 11: Number of Components Classified Under Different Classes

Thus it can be seen that the presented system can be used effectively and efficiently for reusability identification and evaluation of function based object oriented software systems using Mahalanobis metric based pseudo learning algorithm which makes the accuracy of predictions independent of population of classes in the training sample space. The presented tool not only estimates effective reuse of a component for direct or complete reuse but also provides effective estimates of partial reuse or reuse after modification. Thus the presented model (JRPML) developed can be of great use to software practitioners for assessing reusability levels.

VI. Result Comparison with other Approaches

Table 5 below shows results of comparison conducted for JRPML with other existing approaches based on precision, recall (each class), accuracy and classification error values. Comparison results based on accuracy and classification error are shown in Fig. 12 (a) and (b) below.

Table 5: Comparison results with other existing approaches

Attributes	Technique1[22]	Technique2 [39]	Technique3 [25]	JRPML
Tool Support	No	No	No	Yes
Metrics Used	Structural (CK Metric Suite [17])	Functional (CB Metric Suite [10])	Functional (CB Metric Suite [10])	Functional (CB Metric Suite [10] along with Maintainability Index)
Data Mining Approach	k-Means Hybrid	Hierarchical	DBSCAN	LMNN
Support for Estimating efforts of Partial Reuse / Reuse after modification	No	No	No	Yes (Using Maintainability Index)
RESULTS: Accuracy Classification Error Precision Recall	Fig. 12 (a) Fig. 12 (b)	Fig. 12 (a) NA	Fig. 12 (a) Fig. 12 (b)	Fig. 12 (a) Fig. 12 (b)

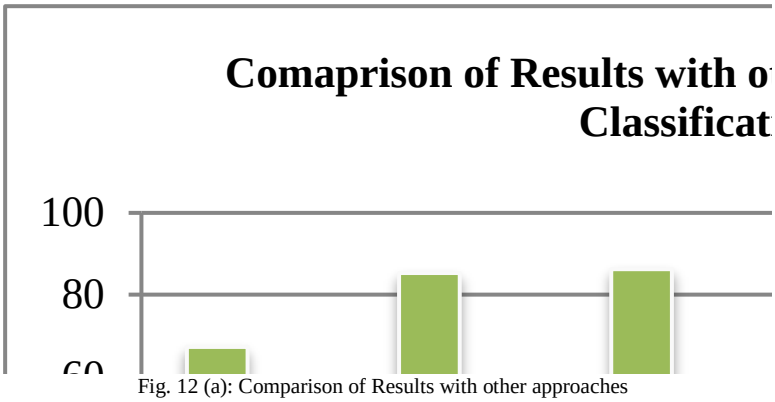

Fig. 12 (a): Comparison of Results with other approaches

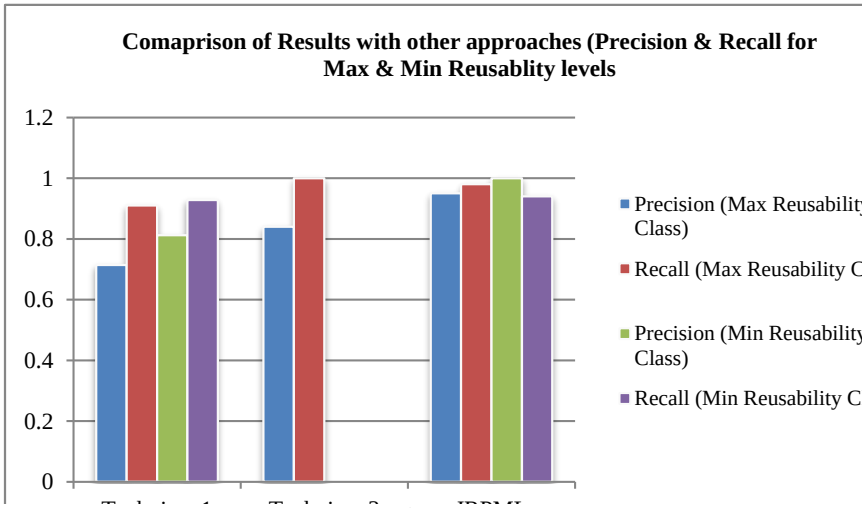

Fig. 12 (b): Comparison of Results with other approaches

From Fig. 12 (a) and (b) it is evident that that results generated by JRPML are much more promising than the results of other existing approaches. JRPML has higher accuracy levels and lower levels of classification error as compared to technique 1, 2 and 3. Unlike JRPML, technique 1 focuses on structural analysis for reusability prediction rather than functional, while techniques 2 and 3 focus on functional aspect but have no metric to assess maintainability of the reusable component. From Fig. 12 (b) it can be seen that the precision recall results for JRPML are better in comparison to technique 1 and almost similar to those of technique 3 for high reusability class. But for low reusability class precision and recall values are 0 for technique 3, thus technique 3 increases the risk of identifying non reusable components as reusable. Hence it can be conclude that JRPML provides better results than other existing approaches.

VII. Application of Proposed Work

The proposed approach supports software practitioners, increasing their throughput by promoting reuse of components via the knowledge of predicted levels of reusability of the newly developed or already available components. The adoption of the presented approach may help industries in strengthening their software reservoirs using the existing inventories while saving greatly on cost and time. Organizations can realize time to market benefits for a new product with this approach. The application of the work may also provide for much better customer satisfaction, helping software developers improve the quality, reliability, maintainability and effectiveness of the software codes.

VIII. Conclusion

In the paper a functional metric and machine learning based model for identifying and evaluating levels of reusability of the components of procedure based object oriented software systems is presented. The model has been implemented as a tool support namely JRPML in Java. The Meta data generated by the functional metrics using maintainability index are analyzed by the proposed algorithm LMNN, where pseudo learning based training is done before prediction for more efficient results. The model used for evaluating reusability levels provides for 100% precision in identifying low reusable components thus reducing risks associated with wrong reuse. For medium and high reusable components the precision values are 83% and 95% respectively. The recall values are 98%, 83% and 94% respectively for high, medium and low reusability levels, thus providing for easy identification of highly reusable components as 98% of them are recognizable by the developed automation tool with 95% precision. From the results generated by our tool JRPML we can see that the tool is able to clearly differentiate between components with ‘HIGH’ and ‘LOW’ reusability values at functional level. The overall accuracy levels provided by the system are 94.2%, and a classification error rate of 5.8%. Hence with the performance results of our tool JRPML, it can be efficiently used by software developers and practitioners as an accurate automated solution for identification and evaluation of reusable components of procedure based object oriented systems.

In future we plan to extend the tool further for reusability analysis and evaluation of other Object Oriented languages like C++ and Python. We will also try to investigate the uses of proposed tool JRPML for extending towards providing more precise classification of components by further increasing the number of classification levels of reusability evaluation.

References

[1] Mili H., Mili F., Mili A. Reusing Software: Issues and Research Directions [J].IEEE Transactions on Software Engineering, 1995, 21(6):528-562.

[2] Mcllroy D. Mass Produced Software Components [C]. Software Engineering Concepts and Techniques NATO Conference on Software Engineering 1968, pp. 88-98.

[3] Singh S., Singh S., Singh G. Reusability of the Software [J]. International Journal of Computer Applications, 2010, 7(14):38-41.

[4] Peters J. F., Pedrycz W. Software Engineering: An Engineering Approach [M]. John Wiley & Sons, NewYork, N Y, 2000

[5] Jalender B., Govardhan A., Premchand P. A Pragmatic Approach to Software Reuse [J]. Journal of Theoretical and Applied Information Technology, 2011, 13(6): 87-96

[6] Cybulski J L. Introduction to Software Reuse [R]. Technical Report TR 96/4 The University of Melbourne Australia, 1996.

[7] Prieto-Diaz R., Freeman P. Classifying software for Reusability [J]. IEEE Software, 1987, 4(1): 6-16

[8] Selby, Richard W. Quantitative Studies of Software Reuse in Software Reusabilty [M]. Addison-Wesley, Reading, MA, 1989.

[9] Chen, Deng-Jyi, Lee P. J. On the Study of Software Reuse Using Reusable C++ Components [J]. Journal of Systems Software, 1993, 20(1):19-36.

[10] Caldiera, Gianluigi, Basili V. R. Identifying and Qualifying Reusable Software Components [J]. IEEE Software, 1991, 24(2):61-70.

[11] Karlsson, Even-Andre, Sindre G., Stalhane T. Techniques for Making More Reusable Components [R]. REBOOT Technical Report, 1992.

[12] Hislop, Gregory W. Using Existing Software in a Software Reuse Initiative [A].In: Proceedings of The Sixth Annual Workshop on Software Reuse, Owego, New York, 1993.

[13] Boetticher G., Srinivas K., Eichmann D. A Neural Net-based Approach to Software Metrics [C]. In: Proceedings of the 5th International Conference on Software Engineering and Knowledge Engineering (SEKE'93) June 1993, San Francisco, CA, pp.271-274.

[14] Torres, William R., Mansur H., Samadzadeh. Software Reuse and Information Theory Based Metrics [C]. In: Proceedings of Symposium on Applied Computing, Kansas City, MO, April 1991, 437-46.

[15] Mayobre, Guillermo. Using Code Reusability Analysis to Identify Reusable Components from the Software Related to an Application Domain [A]. In: Proceedings of Fourth Annual Workshop on Software Reuse, Reston, VA, 1991.

[16] RAPID. RAPID Center Standards for Reusable Software [R]. U.S. Army Information Systems Engineering Command, 1990.

[17] Chidamber S. R., Kemereer C. F. A metrics suite for object oriented design [J]. IEEE Transactions on Software Engineering, 1994, 20(6):476-493

[18] Czibula I. G., Serban G. Heirarchial clustering for Software System Reconstructing [R]. Babes bolyai University, Romania, 2007.

[19] Goel H., Singh G. Evaluation of Expectation Maximization based Clustering Approach for Reusability Prediction of Function based Software Systems [J]. International Journal of Computer Applications, 2010, 8(13):13-20

[20] Kanellopoulos Y., Dimopulos T., Tjortjis C., Makris C. Mining source code Elements for Comprehending Object-Oriented systems and Evaluating Their Maintainability [C]. In: Proceesing of ACM SIGKDD Explorations Newsletter, 8(1), 2006 : 33-40

[21] Manhas S., Sandhu P.S., Chopra V., Neeru N. Identification of Reusable software Modules in Function Oriented Software System using Neural Network Based Technique [J]. World Academy of Science, Engineering and Technology, 2010, 67:823-827

[22] Shri A., Sandhu P. S., Gupta V., Anand S. Prediction of Reusability of Object Oriented Software System using clustering Approach [J]. World Academy of Science, Engineering and Technology, 2010, 67:853-856.

[23] Kumar A. Measuring Software reusability using SVM based classifier approach [J]. International Journal of Information Technology and Knowledge Management, 2012, 5(1): 205-209

[24] Kaur A., Singh R., Cheema, Sandhu P. S. Identification of Reusable Procedure Based Modules using k-NN Approach [C]. In: Proceedings of International Conference on Latest Computational Technologies (ICLCT'12), March 2012, pp.90-93.

[25] Saini J. K., Sharma A., Sandhu P. S. Software Reusability Prediction using Density Based Clustering. 2006, psrcentre.org.

[26] Sandhu P. S., Singh H. A Reusability Evaluation Model for OO-Based software Components [J]. International Journal of Electrical and Computer Engineering, 2006,1(4):259-264

[27] Sandhu P. S., Singh J., Gupta V., Kaur M., Manhas S., Sidhu R. A K-Means Based Clustering Approach for finding Faulty Modules in Open Source software Systems [J]. World Academy of Science, Engineering and Technology, 2010, 72:654-658

[28] Welker K. D. The Software Maintainability Index Revisited [J]. The Journal of Defense Software Engineering, 2001:18-21

[29] McCabe T. J. A Complexity Measure [J]. IEEE Transaction on Software Engineering, 1976, 2(4): 308-320

[30] Halstead, M. H. Elements of Software Science [M]. Elsevier North-Holland, New York, 1977

[31] Mitchell, T. Machine Learning [M]. McGraw Hill. 1997.

[32] Weinberger, K. Q., Blitzer J. C., Saul L. K. Distance Metric Learning for Large Margin Nearest Neighbor Classification [C]. In: Proceedings of Advances in Neural Information Processing Systems (NIPS'06), 2006.1473–1480.

[33] Wikipedia – k-Nearest Neighbour Algorithm. http://en.wikipedia.org/wiki/K-nearest_neighbor_algorithm

[34] Mahalanobis P. C. On the generalised distance in statistics [C]. In: Proceedings of the National Institute of Sciences of India, 1936, 49–55.

[35] Wikipedia – Semidefinite Programming. http://en.wikipedia.org/wiki/Semidefinite_programming

[36] Dimitrov E., Wipprechet M., Schmietendorf A. Conception and Experience of Metric-Based Software Reuse in Practice [A]. In: proceedings of International Workshop on Software Measurements, Canada, 1999.

[37] Wikipedia-Confusionmatrix. http://en.wikipedia.org/wiki/Confusion_matrix#cite_note-0

[38] Code Metric Values. http://msdn.microsoft.com/en-us/library/bb385914.aspx

[39] Verma P., Mahajan M., Gupta M. Hierarchical Clustering Approach for Modeling of Reusability of Function Oriented Software Component. ISEMS, http://isems.org.

Authors' Profiles

Surbhi Maggo: has done Masters of Technology and Bachelor of Technology from Jaypee Institute of Information Technology, India in Computer Science & Engineering and her area of interest is Software Engineering, Software Testing, Data Mining and Machine Learning. Currently she is working in a research and development company in India.

Chetna Gupta: She is Assistant Professor at Jaypee Institute of Information Technology, India. She obtained her Doctorate in the area of Software Testing. She also holds a Masters of Technology and a Bachelor of Engineering degree in Computer Science and Engineering. Her areas of interest are Software Engineering, Requirement Engineering, Software Testing, Software Project Management, Data Structures, Data Mining and Web Applications. She has many publications in international journals and conferences to her credit.

How to cite this paper: Surbhi Maggo, Chetna Gupta,"A Machine Learning based Efficient Software Reusability Prediction Model for Java Based Object Oriented Software", International Journal of Information Technology and Computer Science(IJITCS), vol.6, no.2, pp.1-13, 2014. DOI: 10.5815/ijitcs.2014.02.01