Assessing Impact of Class Change by Mining Class Associations

Anshu Parashar and Jitender Chhabra
Department of Computer Engineering, National Institute of Technology, India

Abstract: *Data mining plays vital role in data analysis and also encompasses immense potential of mining software engineering data to manage design and maintenance issues. Change impact assessment is one of the crucial issues in software maintenance. In Object Oriented (OO) software system, classes are the core components and changes to the classes are always inevitable. So, OO software system must support the expected changes. In this paper, to assess impact of change in the class, we have proposed changeability measures by mining associations among the classes. These measures estimate a) change propagation by identifying its ripple effect; b) change impact set of the classes; c) changeability rank of the classes and d) class change cost. Further, we have performed the empirically study and evaluation to analysis our results. Our results indicate that by mining associations among the classes, the development team can effectively estimate the probable impact of the class change. These measures can be very helpful to perform changes to the classes while maintaining the software system.*

Keywords: *Mining software engineering data, object oriented system development, change propagation, change impact.*

Received September 7, 2015; accepted February 21, 2016

1. Introduction

Designing a changeable software system allow developers to effectively develop and maintain it. As we know that, a class is a core component of an object oriented software system and the whole application revolves around classes and packages. Class mostly works in association where it participates with other classes in order to achieve a nearly similar goal or to implement the common functionality. Due to this collaborative nature of classes they require proper attention while changing them. Change Impact Assessment (CIA) is a very crucial task during the maintenance. A software system is said to be more maintainable if its components (e.g., classes) can be changed or modified easily without much efforts [8, 16] Research into Change Impact Assessment (changeability assessment) consists of changeability predictors based by measuring and predicting association (coupling) among the software artifacts (i.e., classes, packages) [28]. Jabangwe *et al*. [19] extensively investigated the link between coupling measures and external quality attributes. In their empirical study, they also mapped changeability and its proxies (change impact, change-proneness) to maintainability. Measurement and prediction of association (coupling) can be carrying out based on structural or design analysis of software system [22, 28, 30]. Two classes can be considered as change-coupled if they are structurally (design) coupled. The software engineering community has recognized that the excessive change-coupling among classes can lead to serious problems during development as well as maintenance. Fowler described excessive change-coupling as "when every time you make a kind of change, you have to make a lot of little changes all over the place, they are hard to find, and it's easy to miss an important change" [15]. From the software evolution perspective, classes that are more design coupled will have more chances to be changed together. There may be some classes that are changed together but they may not be design coupled. This may be due to the poor design. If a change is made to a class C_i, then by exploring the coupling behavior of classes it can be measured easily that how far this change can propagate its impact. Classes that are coupled with C_i i.e., import its functionality, have ample chance to be candidates that can be affected due to change in C_i.

Mining software engineering data should be presumed as effective instrument for developers/maintainers and existence of coupling at class level should be analyzed with the help of the data mining to measure the software quality [4, 25, 26, 32]. Especially, association mining can be very useful to identify the highly associated (interdependent) classes of a software system by mining information about the dependencies existing among them. If a class C_i is being used by other classes then it is probably obvious that a change in C_i can be propagated to its associated (dependent) classes. So, during the software development/maintenance, class change impact assessment by mining association among the classes can be a vital aid for development/maintenance team. Hence to address this need, in this paper, associations among the core components i.e., classes have been mined and their behavior of coupling are explored to assess their changeability (e.g., impact of class

changes) for the future. In order to assess impact of changes in the classes, we have defined some changeability measures on the basis of associations (interdependency) between them. We have mined all possible associations among classes (direct as well as indirect). The reason behind this is that we want to know, how far change in the class can propagate. In our work, we have targeted to mine class coupling data of object oriented software system. In the context of this paper, changeability or change impact of a class has been considered as a function of the propagation (ripple effect) of the change on other classes.

The primary contributions of this paper are:

1. We have estimated the change propagation of a class C_i by identifying its ripple effect. For this, a metric named as *Change Propagation Index* of a Class-*ChPI(C_i)* has been defined. *ChPI(C_i)* has been computed by mining associations among classes.
2. We have ranked the classes as per their changeability in terms of percentage of classes that can be affected due to change in a class C_i. A metric named as *Class Changeability Rank- CChR(C_i)* has been defined.
3. In order to predict impact of the class change, we have defined a metric named *as Change Impact class set* of a class C_i- *ChImpactSet(C_i).*
4. Further, the *Change Cost* of a Class C_i- *$Cost_{change}$(C_i)* has been computed. For this, weighted coupling between each pair of classes *WDC(C_i,C_j)* are utilized to quantify the change cost as a function of extent of association (proximity) a of class C_i with rest of classes which import its functionality.

The proposed measures mentioned above will be very helpful and provide assistance to the developer/maintainer while accommodating the change. Maintainer will know;

1. Quantitatively how far a change to a class can be propagated.
2. Probable classes that can be more affected due to the class change.
3. Change cost of the class and d) Changeability rank of the classes of software system.

The rest of the paper is organized as follows. The section 2 discusses the related literature. Section 3 describes extraction of class associations and also defined proposed changeability measures. Section 4 describes the empirical results and validation section 5 concludes the paper and presents scope of the future work.

2. Related Work

The efforts for maintaining the software system contribute much in the overall efforts of the software development. Software maintainability can be estimated by knowing quality attributes e.g., changeability. In the Object Oriented (OO) context, various metrics had been proposed and evaluated [5, 7, 9, 12, 17, 18, 20] and almost all these had been concluded the strong relationship between design metrics and maintenance efforts. Much research has been carried out to evaluate the influence of size of change component, their structural properties on change management [6, 10, 29]. Association (coupling) between classes reflects valuable quality information about the design of software and enables the software designer to plan various important issues of the software engineering like reusability, maintainability, change propagation, fault prediction etc., [17]. Some empirical studies have demonstrated that object-oriented coupling metrics are correlated with various aspects of maintainability [10, 11, 17]. Benestad *et al*. [7] tried to identify cost drivers of software evolution. Briand *et al*. [11] proposed UML model-based impact analysis. Further, Vanya et al investigated co-evolving entities and discuss how interactive visualizations can support the process of analyzing structural issues [30].

Estimation of software quality using data mining techniques helps software managers to monitor and control potential quality issues over the time [3, 27, 33]. Fayyad *et al*. [14] described it as *"*Knowledge Discovery in Databases (KDD) is the non-trivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in data*"*. Using mining, lot of research has also been carried out in the fields of system evolution, comprehension and remodularization [24, 26, 32]. Frequent item sets and association rule mining introduced by Agrawal et al. can play essential role for examine data sets. They have introduced algorithm named Apriori, described the concepts of Association Rule, Support and Confidence [1, 2]. Nowadays data mining algorithms are integrated with the software development models to assist in different software engineering processes like testing, debugging, maintaining and also to improve productivity along with quality [19, 22, 28, 31]. Several researcher studied changeability as CIA [22, 28] to determine impact of proposed changes on components of software systems. Sun *et al.* [28] used formal concept analysis for assessing change impact. From literature, we have observed that, mining software measurement data can be handy to better understand software development and improvise various software engineering tasks. So, in our work, we have targeted to mine class coupling data of object oriented software system.

3. Proposed Measures for Change Impact Assessment

The aim of this paper is assess the impact of changes in the classes by mining association among them. The primary objective of our approach is to estimate the

changeability of the class in terms of its change impact assessment. Here, we have proposed measures to predict

1. Change propagation of the class.
2. Changeability of the class.
3. Change cost for the class.
4. Change impact set of the class.

In this section, firstly we have described the process of extracting class associations (couplings) data, secondly, we have described two suitable representation of this data to further mine it for the computation of proposed measures; and finally, we have defined and describe the proposed change impact assessment measures.

3.1. Data Extraction

As a prerequisite for our objectives, weighted direct associations (couplings) between the classes have been computed. It tells how much a class depends on other classes. After this, it is suitably represented as weighted coupling matrix (Direct) M_D, and then M_D is repeatedly multiplied to find out all possible associations direct as well as indirect among the classes in the form of M_{D+I}. M_{D+I} is further used to compute the proposed measures. In next sub-sections we have described it in detail.

3.1.1. Collection of Class Coupling Data

In view of predicting change impact, we decided to use weighted coupling between classes as proposed by Gui and Scott [17]. Here, java based software systems are our underlying systems for our analysis. Any Java based system can be regarded as a directed graph in which the vertices correspond to its classes and the edges correspond to direct associations between them. Consider an example package (i.e., Package) PKG comprising of set of N classes ($N=8$) and its $Class\_Set(PKG)=\{C_0, C_2, \ldots, C_7\}$ represented as a graph (Figure 1). Here, each vertex contain name of the class, its members count $(|M_i|)$ and the number of all members of other classes invoked by C_i is $|X_i|$. It indicates the extent to which class C_i depends upon other classes. An edge between class C_i and C_j represents that class C_i uses some members of class C_j and the count of these members is given by $|X_{i,j}|$ which is the number of members of class C_j invoked by class C_i and this $|X_{i,j}|$ is used as the weight of the edge between C_i and C_j. Further, the weighted direct coupling $WCD(C_i,C_j)$ [17] between class C_i and C_j is computed as per the formula 1 and mentioned in Figure 2.

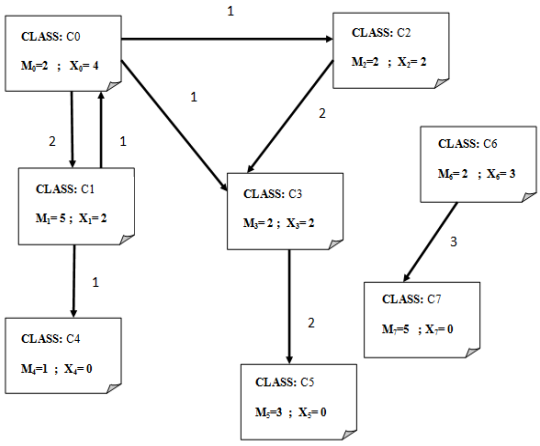

Figure 1. Class coupling graph of an example package PKG.

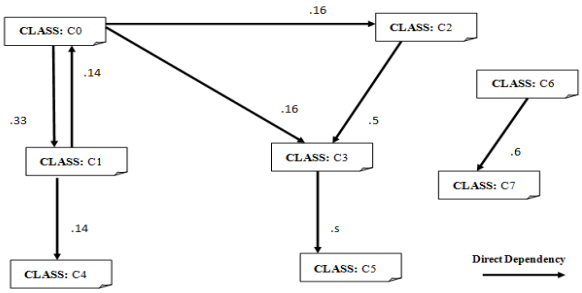

Figure 2. Direct coupling graph with $WCD(C_i,C_j)$ weights of package PKG.

$$WCD(C_i, C_j) = \frac{|X_{i,j}|}{|X_i| + |M_i|} \quad (1)$$

After computing the direct coupling weights, this data should be represented in some suitable intermediate representation. For this, we choose to represent class association data in following two forms, which are suitable for our measures.

3.1.2. Weighted Coupling Matrix Representation

Here, each class is represented as weighted coupling vector $C\_V(C_i)=[x_{i1}, x_{i2}, \ldots, x_{iN}]$, $i = 1,\ldots,N$, where N is the same as the number of classes in the package. Value of each weight x_{ij} is the actual coupling measure between each pair of classes C_i & C_j. In this way of representation, coupling between class pairs is calculated based on the extent of coupling between classes on the scale of 0 to 1. If two classes are highly coupled, then their coupling is represented by a value close to 1. After having weighted coupling vectors of all N classes, it is further represented in the form of as matrix M_{D+I} (all possible association direct as well indirect). To calculate all possible associations between the classes, we use matrix multiplication. We multiply the weighted coupling matrix (M^0) with itself which gives (M^1), to find all the indirect couplings of level one (means the indirect couplings through a single class as $C_i \rightarrow C_j \rightarrow C_k$). Similarly, ($M^1$) is multiplied with ($M^0$) to find all the indirect couplings of level two (means the indirect couplings through two

classes as $C_i \rightarrow C_j \rightarrow C_k \rightarrow C_m$). The procedure is repeated N (total no of classes of package) times to calculate the indirect dependencies of all possible levels. Then the final dependence matrix M_{D+I} (Table 1) containing all possible dependencies (direct as well as indirect) is computed by formula 2:

$$M_{D+I} = \sum_{k=0}^{N} M^k \qquad (2)$$

M^0 is the identity matrix, where for each *i*, $(M^0)_{i,i}=1$, it represents the dependency of each class on itself. The purpose of computing the indirect coupling also is that along with direct coupling, indirect couplings also affect the change propagation, change cost as well as the frequently coupled class set.

Table 1. Weighted coupling matrix M_{D+I} of package PKG.

Classes	C_0	C_1	C_2	C_3	C_4	C_5	C_6	C_7
C_0	0.131	0.043	0.020	0.031	0.006	0.015	0	0
C_1	0.018	0.131	0.002	0.004	0.018	0.002	0	0
C_2	0	0	0.125	0.062	0	0.031	0	0
C_3	0	0	0	0.125	0	0.062	0	0
C_4	0	0	0	0	0.125	0	0	0
C_5	0	0	0	0	0	0.125	0	0
C_6	0	0	0	0	0	0	0.125	0.075
C_7	0	0	0	0	0	0	0	0.125

3.1.3. Export Coupling Set Representation

The collected class association (coupling) data for each class can be treated as class coupling transaction. The idea of this representation is similar to the market-basket analysis approach used in data mining. The details of this can be found from Agrawal *et al*. [2]. Here, class name C_i is assumed as class Coupling Transaction id- $C\_T_{id}(C_i)$ and the set of classes depends on C_i are recorded as its Export Coupling Transaction Set- $EC\_TSet(C_i)$. For example package i.e., PKG, all possible coupling transactions among classes are represented in Figure 3. If an edge is coming from class C_j to C_i, it means C_i exports some of its functionality from C_j. Irrespective of coupling weight; we are considering whether two classes are coupled or not i.e., binary weights. Similarly, classes from where the edges are coming to C_i are included in its $EC\_TSet(C_i)$. $EC\_TSet(C_0)=\{C_0,C_1\}$ means class C_0 is only used by class C_1 and this set also include the class C_0 itself. Table 2 shows coupling transaction of all classes of example package *PKG*.

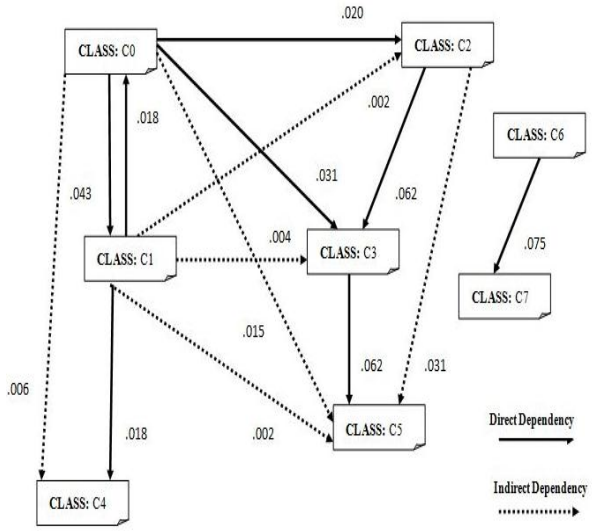

Figure 3. Weighted coupling graph of all possible coupling of package PKG.

Table 2. Coupling transactions for classes of package PKG.

C_TID	EC_TSet(C_i)
C_0	C_0 , C_1
C_1	C_0 , C_1
C_2	C_0 ,C_1,C_2
C_3	C_0 , C_1 , C_2 , C_3
C_4	C_0 , C_1 , C_4
C_5	C_0 , C_1 , C_2 , C_3 , C_5
C_6	C_6
C_7	C_6 , C_7

3.1.4. Utilization of Above Representations

The appropriate representation of collected class associations is the prerequisite for the computation of our proposed measures. The first representation, weighted coupling matrix provides actual coupling weights. This information can also be represented through the binary coupling weights but we are using actual coupling weights instead of binary weights. The reason behind this is, the binary weighted only gives whether coupling exists or not and it does not reflect the extent of coupling between classes. The weights estimate the coupling quantitatively. In our work, we are using this scheme to compute the change cost for the classes. The inspiration behind using this representation is that we can have the change cost based on extent of coupling between classes. In the second representation i.e., export coupling set, export coupling of each class is recorded as transactions i.e. $EC\_TSet(C_i)$ (as purchase transaction in market basket analysis). So, we can have *N* (no's of classes in package *P*) coupling transactions $\{EC\_TSet(C_i)....EC\_TSet(C_N)\}$. This representation is well proven and widely used to mine the associations between the data items though algorithms like Apriori, Frequent Pattern (FP), FP tree [1, 2]. We choose this to further apply Apriori algorithm to produce frequent coupling set as well as for the computation of change propagation index. Both representation are conveying the similar information but in different ways and for different utilizations as described above. Example

package PKG has been used to describe the proposed changeability measures and to demonstrate their computation. The next section describe the measures i.e., metrics and their computation.

3.2. *Metrics Definition and Computation*

a) Class Change Propagation Index-CChPI(C_i): CChPI(C_i) quantifies how far the change made to class C_i can be propagated to rest of the classes of the package P. CChPI(C_i) will be measured by mining association of class C_i with rest of the classes. It measures the degree of propagation of change that is made to class C_i by exploring, how the other classes of package P are associated with C_i i.e., EC_ TSet(C_i). The reason behind this is, if a class C_i is being used by the set of classes {C_x, C_y,C_z}, then it is obvious that any changes made to class C_i can produce ripple effect to these classes. So, Change Propagation Index of a Class C_i-CChPI(C_i) is defined as (Equation (3)):

$$ChPI(C_i) = \sum_{j=1, j\neq j}^{N} Prox(C_i, C_j) \quad (3)$$

Where, *N* is the number of classes in a package *P* and *Prox(C_i,C_j)* measures the association of a class C_i with C_j in term of how often the class C_j is coupled with class C_i as described in formula 4. It indicates the extent of proximity of class C_j with C_i with respect to the all classes in the package *P*. The *Prox(C_i,C_j)* is to be calculated in the same way as the association evaluation between two items i.e., *confidence($X \rightarrow Y$)* [1, 2].

$$Prox(C_i, C_j) = \frac{|C_i \cap C_j|}{|C_i|} \quad (4)$$

Here $C_i \cap C_j$ is measured through class coupling transactions (Table 2). It gives the number of containing both C_i and C_j. As per the coupling behavior of the classes, it tells how many times C_i and C_j are used together. $|C_i|$ gives count of transactions in which C_i exists. It tells the number of classes of package *P* using C_i. Further, propagation of change then also be computed as the percentage of classes that can be affected due to change in C_i –*i.e., %CChPI(C_i)* as per Equation 5. In other words it gives percentage of classes in which change can produce its ripple effect.

$$\%CChPI(C_i) = (CChPI(C_i)/N)*100 \quad (5)$$

For a class, it is always desirable to have low value of change propagation index because it is always difficult to accommodate the change in the classes with high change propagation indices. For example package (i.e., *PKG*), *%CChPI(C_i)* of each class of the PKG have been computed as shown in Table 3 (according to the coupling transactions tabulated in Table 2).

b) Class Changeability Rank- *CChR(C_i):* It can be measured by exploring the *CChPI(C_i)* of a class in terms of %of classes that can be affected due to change in C_i. Here changeability of classes of package *P* is categorized as *Good*, *Moderate* and *Critical* on the basis of following rules as (Equation (6)):

$$\begin{aligned} &if(\%CChPI(C_i) \leq \theta_1) \text{ then } CChR(C_i) = Good \\ &elseif(\%\ CChPI(C_i) \leq \theta_2) \text{ then } CChR(C_i) = Moderate \\ &elseif(\%CChPI(C_i) > \theta_2) \text{ then } CChR(C_i) = Critical \end{aligned} \quad (6)$$

The value of thresholds θ_1 and θ_2 are to be decided by the development team on the basis of how much association (coupling) among the classes are permissible to rank their changeability as Good, Moderate and Critical.

Table 3. CChPI(C_i) %CChPI(C_i) and CChR(C_i), for classes of package PKG.

Class	CChPI(C_i)	%of class can be affected due to change in C_i (%CChPI(C_i))	CChR(C_i)
C_0	2.15	26.9%	Moderate
C_1	2.15	26.9%	Moderate
C_2	2.99	37.3%	Moderate
C_3	3.5	43.8%	Critical
C_4	2	25%	Moderate
C_5	5	62.5%	Critical
C_6	0.5	6.3%	Good
C_7	1	12.5%	Good

The classes having *CChR(C_i)=Good* are said to be easily changeable/ maintainable or ready to be changed. The classes having *CChR(C_i)=Moderate* are supposed to be difficult to change /maintain or are not easily changeable as compare to the classes having changeability rank as *Good*. The classes having *CChR(C_i)=Critical*, It means these classes are highly coupled and more difficult to change. These classes will require more attention whenever a change request comes for these classes because they are adversely changeable. For example package (i.e., *PKG*), *CChR(C_i)* of each class of the *PKG* have been computed as shown in Table 3 (assuming θ_1=20 and θ_2=40).

c) Change Impact Set for a class- ChImpactSet(C_i): Whenever a change to a class occurs, it becomes important to identify the set of classes which will be impacted due to this change. Maintainer has to ensure the correctness of the application after the change is accommodated. To ensure this, it is required to properly address or reflect this change to the set of change impact classes. Once a given class C_i is subject to change, *ChImpactSet(C_i)* indicates which other classes in the package *P* will be affected by this change. Therefore, one can analyze how exactly the interdependency among the classes of a package affects its changeability. So *ChImpactSet(C_i)* can be measured as:

$$ChImpactSet(C_i) = \{C_j \mid Prox(C_i, C_j) \geq Prox_{th}, \text{ where } j \neq i\} \quad (7)$$

It means, change impact set of class C_i includes set of

all classes (*except class* C_i) whose proximity (association) with C_i is greater than or equal to a specified proximity threshold $Prox_{th}$. Here, $Prox_{th}$ indicates the minimum permissible proximity (association) between the classes and it is to be decided by project team. Table 4 shows the *ChImpactSet*(C_i) of all the classes of package *PKG*, considering proximity threshold $Prox_{th}$ *=0.40*.

Table 4. Change impact set-*ChImpactSet*(C_i) of classes of package. PKG.

Class	ChImpactSet(C_i)
C_0	C_1 , C_2
C_1	C_0 , C_2
C_2	C_0 , C_1 , C_3
C_3	C_0 , C_1 , C_2 , C_5
C_4	C_0 , C_1
C_5	C_0 , C_1 , C_2 , C_3 , C_4
C_6	C_7
C_7	C_6

d) Change Cost of class-$Cost_{Change}(C_i)$ The motive behind capturing the associations among the classes is to know the degree to which a change in any class causes a change to other classes in the package, either directly or indirectly. Our idea of computing the change cost is somewhat similar to [23]. The two significant differences are the level of computation and weighting criteria of the coupling. So, firstly if we talk about the level of computation, they have computed the change cost at system level as a function of the coupling among packages. Instead of this, we are computing the change cost at package level as a function of the coupling among classes. Secondly, they consider direct or indirect coupling as binary (1/0) instead of this we choose weighted coupling between classes. So, the weighted coupling matrix M_{D+I} derived through the procedure mentioned in section 3.1 is considered to compute the change cost of each class $Cost_{Change}(C_i)$ of package. So, the change cost of class C_i– $Cost_{Change}(C_i)$ is calculated as (Equation (7)) average of sum of coupling weights in a column i of M_{D+I}. It gives the weighted fan in value of class C_i. For a class C_i, it is always desirable to have low $Cost_{Change}(C_i)$. A class with high $Cost_{Change}(C_i)$ indicates extent of dependency of other classes of package *P* on C_i is high.

$$Cost_{Change}(C_i) = \sum_{j=1}^{N} M_{D+I}[j,i] \qquad (8)$$

For example package *PKG*, $Cost_{Change}(C_i)$ for all classes of *PKG* is shown below in Table 5.

Table 5. Change cost $Cost_{Change}(C_i)$of classes of package PKG.

Class	$Cost_{Change}(C_i)$
C_0	0.15
C_1	0.17
C_2	0.15
C_3	0.22
C_4	0.15
C_5	0.24
C_6	0.13
C_7	0.20

In next section we have demonstrated the empirical evaluation of proposed measures.

4. Empirical Results and Evaluation

To study the results of proposed measures, we have applied them on the classes of Java Development Kit (JDK) packages AWT, *IO* and LANG. To properly evaluate our proposed metrics we chose classes that have all levels of coupling like highly coupled, less coupled and not coupled. The packages and their considered classes are mentioned in Table 6. It describes the package number, name and their set of classes i.e., $C_{j,i}$, where each $C_{j,i}$ represents the i^{th} class of the package *j*. Changeability measures are computed and tabulated in Tables 7, 8, 9, and 10.

Table 6. Packages (*AWT*, *LANG* and *IO*) and their class set.

Package No.	Package Name	Classes in Packages
1	AWT	$C_{1,0}$-alphacomposite, $C_{1,1}$- awteventmulticaster, $C_{1,2}$- cardlayout, $C_{1,3}$-checkbox, $C_{1,4}$- ckeckboxmenuitem, $C_{1,5}$-container, $C_{1,6}$-panel, $C_{1,7}$-component, $C_{1,8}$-menuitem, $C_{1,9}$- menucomponent
2	LANG	$C_{2,0}$-system, $C_{2,1}$-string, $C_{2,2}$-math, $C_{2,3}$-object, $C_{2,4}$-number, $C_{2,5}$-package
3	IO	$C_{3,0}$-bufferedinputstream,$C_{3,1}$-bufferreader,$C_{3,2}$-bytearray inputstream, $C_{3,3}$-chararrayreader, $C_{3,4}$-objectinputstream, $C_{3,5}$-bufferedoutputstream,$C_{3,6}$-filterinputstream,$C_{3,7}$-filteroutputstream , $C_{3,8}$-reader , $C_{3,9}$- inputstream

Here, we briefly mention the findings derived from these empirical study and results. For package AWT, results (Table 7) show that CChPI of class component is the highest among all classes of AWT.

It indicates that class component is extensively used by other classes as compared to the rest of classes of AWT. Any change in class component should be properly addressed and should be reflected to all the classes of its change impact class set (Table 8) i.e., cardlayout, ckeckboxmenuitem, container, panel. CChPI of classes alphacomposite, awtevent-multicaster are nil. It shows changes in these classes will not affect any other classes of AWT.

Table 7. CChPI(C_i) %CChPI(C_i) and CChR(C_i), for classes of packages AWT, LANG and IO.

Package-AWT				Package-LANG				Package-IO			
Class	CChPI($C_{j,i}$)	%CChP($C_{j,i}$)	CChR(C_i)	Class	CChPI($C_{j,i}$)	%CChP($C_{j,i}$)	CChR(C_i)	Class	CChPI($C_{j,i}$)	%CchP($C_{j,i}$)	CChR(C_i)
$C_{1,0}$	0	0	Good	$C_{2,0}$	2.5	41.6	Critical	$C_{3,0}$	0	0	Good
$C_{1,1}$	0	0	Good	$C_{2,1}$	2.5	41.6	Critical	$C_{3,1}$	1	10	Good
$C_{1,2}$	1.65	16.5	Good	$C_{2,2}$	2.5	41.6	Critical	$C_{3,2}$	1	10	Good
$C_{1,3}$	1	10	Good	$C_{2,3}$	4	66	Critical	$C_{3,3}$	1	10	Good
$C_{1,4}$	1.4	14	Good	$C_{2,4}$	2.5	41.6	Critical	$C_{3,4}$	1	10	Good
$C_{1,5}$	2.5	25	Good	$C_{2,5}$	2.5	41.6	Critical	$C_{3,5}$	0.5	5	Good
$C_{1,6}$	2.5	25	Good					$C_{3,6}$	0	0	Good
$C_{1,7}$	4	40	Moderate					$C_{3,7}$	1	10	Good
$C_{1,8}$	1	10	Good					$C_{3,8}$	2	20	Good
$C_{1,9}$	1	10	Good					$C_{3,9}$	2	20	Good

Table 8. Change Impact set of classes of packages *AWT.LANG* and *IO*(*Prox$_{th}$= 0.60).*

Package-AWT		Package-LANG		Package-IO	
Proximity Threshold - Prox$_{th}$=.60					
Class	**ChImpact($C_{j,i}$)**	**Class**	**ChImpact($C_{j,i}$)**	**Class**	**ChImpact($C_{j,i}$)**
$C_{1,0}$	{$C_{1,0}$}	$C_{2,0}$	{$C_{2,0}$ }	$C_{3,0}$	{$C_{3,0}$}
$C_{1,1}$	{$C_{1,1}$}	$C_{2,1}$	{$C_{2,1}$}	$C_{3,1}$	{ $C_{3,1}$}
$C_{1,2}$	{$C_{1,2}$;$C_{1,5}$}	$C_{2,2}$	{ $C_{2,2}$}	$C_{3,2}$	{ $C_{3,2}$}
$C_{1,3}$	{$C_{1,3}$;$C_{1,4}$}	$C_{2,3}$	{$C_{2,0}$;$C_{2,1}$;$C_{2,2}$;$C_{2,3}$; $C_{2,4}$;$C_{2,5}$}	$C_{3,3}$	{$C_{3,0}$}
$C_{1,4}$	{$C_{1,4}$}	$C_{2,4}$	{ $C_{2,4}$}	$C_{3,4}$	{$C_{3,4}$}
$C_{1,5}$	{$C_{1,2}$; $C_{1,5}$}	$C_{2,5}$	{$C_{2,5}$}	$C_{3,5}$	{ $C_{3,5}$}
$C_{1,6}$	{$C_{1,4}$; $C_{1,6}$ }			$C_{3,6}$	{$C_{3,6}$}
$C_{1,7}$	{$C_{1,2}$; $C_{1,4}$; $C_{1,5}$;$C_{1,6}$;$C_{1,8}$}			$C_{3,7}$	{$C_{3,7}$}
$C_{1,8}$	{$C_{1,8}$}			$C_{3,8}$	{$C_{3,1}$; $C_{3,3}$;$C_{3,8}$}
$C_{1,9}$	{$C_{1,8}$; $C_{1,9}$}			$C_{3,9}$	{$C_{3,2}$;$C_{3,4}$;$C_{3,9}$}

ChImpactSet(C_i) of these classes are also null and thus the same observation is true here as well. The classes of package *LANG* seem to be adversely coupled (Table 7).

Table 9. Change cost for classes of packages AWT, LANG and IO.

Package-AWT		Package-LANG		Package-IO	
Class	**Cost$_{Change}$($C_{j,i}$)**	**Class**	**Cost$_{Change}$($C_{j,i}$)**	**Class**	**Cost$_{Change}$($C_{j,i}$)**
$C_{1,0}$	0.1	$C_{2,0}$	0.17	$C_{3,0}$	0.10
$C_{1,1}$	0.1	$C_{2,1}$	0.17	$C_{3,1}$	0.10
$C_{1,2}$	0.1	$C_{2,2}$	0.17	$C_{3,2}$	0.10
$C_{1,3}$	0.11	$C_{2,3}$	0.41	$C_{3,3}$	0.10
$C_{1,4}$	0.1	$C_{2,4}$	0.17	$C_{3,4}$	0.10
$C_{1,5}$	0.17	$C_{2,5}$	0.17	$C_{3,5}$	0.10
$C_{1,6}$	0.1			$C_{3,6}$	0.10
$C_{1,7}$	0.39			$C_{3,7}$	0.11
$C_{1,8}$	0.13			$C_{3,8}$	0.12
$C_{1,9}$	0.13			$C_{3,9}$	0.12

Table 10. Precision, Recall and Cosine Similarity for computed *ChImpactSet($C_{j,i}$)* (*Prox$_{th}$* = 0.60).

Package-AWT				Package-LANG				Package-IO			
Proximity Threshold - Prox$_{th}$=.60											
ChImpact($C_{j,i}$)	**Precision(%)**	**Recall(%)**	**Cosine-Sim**	**ChImpact($C_{j,i}$)**	**Precision(%)**	**Recall(%)**	**Cosine-Sim**	**ChImpact($C_{j,i}$)**	**Precision(%)**	**Recall(%)**	**Cosine-Sim**
$C_{1,0}$	100	100	1	$C_{2,0}$	100	100	1	$C_{3,0}$	100	100	1
$C_{1,1}$	50	100	1	$C_{2,1}$	100	100	1	$C_{3,1}$	100	100	1
$C_{1,2}$	100	100	0.70	$C_{2,2}$	100	100	1	$C_{3,2}$	100	100	1
$C_{1,3}$	100	100	1	$C_{2,3}$	100	100	1	$C_{3,3}$	100	100	1
$C_{1,4}$	100	100	1	$C_{2,4}$	100	100	1	$C_{3,4}$	100	100	1
$C_{1,5}$	100	100	1	$C_{2,5}$	100	100	1	$C_{3,5}$	100	100	1
$C_{1,6}$	50	100	1					$C_{3,6}$	100	100	1
$C_{1,7}$	100	100	1					$C_{3,7}$	100	100	1
$C_{1,8}$	100	100	0.70					$C_{3,8}$	100	100	1
$C_{1,9}$	100	100	1					$C_{3,9}$	100	100	1

All classes are having high change propagation index and all come under the critical category of changeability. The change impact set of all classes itself suggests the same. As far as *IO* package is concerned it appears to be well designed and its classes are loosely coupled (Table 7). The change propagation indices of all classes are very low. It means the change in the classes of IO will produce very less impact on other classes. Further, Table 9 shows the estimated change cost of each class as a function of the extent of the change propagation. It also reflects the same behavior of changeability of classes. For example, here we can see that change cost of class component of package AWT is highest among all the classes of AWT. The similar observation is also derived from CChPI of the class component i.e., it is the highest among all the classes of AWT. Further it can be observed that the change cost of all classes of package IO is low. Whenever a change comes for the class maintainer makes a query or explore *ChImpactSet(C_i)* to know what are the most likely set of classes that will be affected due to change in C_i. The Table 8 shows *ChImapctSet* of classes of packages Abstract Window Toolkit (AWT), (Language) LANG and *IO* by assuming *Prox$_{th}$ 0*.60.

To analyze the correctness of resultant change impact set *ChImapctSet(C_i)* computed by our approach, precision, recall and cosine similarity measures [13, 17] are computed and tabulated in Table 10. These are widely used in data mining and information retrieval tasks [21]. In our case, precision gives the fraction of the predicted change impact classes that are relevant, while recall gives the fraction of the relevant change impact classes that are predicted. For all three java packages, both these measures analyze the predicted change impact set of classes and actual class change impact set. For the classes of all three java packages, here *ChImpactSet(C_i)* is the predicted class change impact set for each class computed by the above proposed measures and actual class change impact set *EC_TSet(C_i)* reflects the set of classes that are highly design coupled with class C_i. The reason behind assuming *EC_Tset(C_i)* as actual class change impact set is, for any change in C_i, classes in *EC_TSet(C_i)* are the most likely candidate classes that can be affected. In order to know the correctness of the predicted change impact set, a measure of cosine(Cosine) similarity [13] between predicted *ChImpactSet(C_i)* and most likely class change impact i.e., *EC_TSet(C_i)* is computed for the three java packages. The comparison of computed precision, recall measures and cosine similarity measures is shown in Table 10. It can be clearly observed that the prediction of *ChImpactSet(C_i)* is satisfactory. As far as cosine similarity is concerned, it measures the similarity between predicted and actual impact sets on the scale of 0 to 1. The *cosine-sim* close to 1 indicates high similarity and 0 indicates very less similarity between two sets. Our results show that predicted change impact classes are very close to actual ones. As was the case with precision and recall, it indicates that our proposed measures predict nearly

accurate set of classes that can be highly affected due to change in the class under consideration.

5. Conclusions and Future Scope

Mining software engineering data can assist software development tasks like maintenance. Measuring class associations first and then, mining association pattern among them definitely become a useful aid to the maintenance team to improve the change progression of classes. In this paper, we have proposed measures to assess the impact of class change using association mining. We have proposed measures for estimating propagation of change made at class level in terms of the change propagation index, rank of changeability, change impact set and the change cost. These measures can be helpful at the time of development as well as maintenance, especially during carried out changes in certain classes. For any change in a class, we also predict the set of classes that can be affected due to that change. The proposed measures are defined, demonstrated through an example package and evaluated on JDK packages. During empirical study and evaluation, it has been found that these measures are useful predictors and also produce results as per expectations. These measures are very helpful for developer as well as maintainer to effectively reflect the changes and also help in minimizing their efforts. Maintainer can know classes with high change propagation index and change cost and pay more focus on these classes as they can increase maintenance costs. Identification of these classes enables developers to properly review change to these classes through proper testing with their dependents. Further, the classes having high change propagation index and change cost can also be considered as candidates for refactoring or restructuring. Moreover, change impact set for a class also help to localize the changes. In future, we will aim to devise data mining based software quality prediction model by mining different artifacts produced during software development life cycle. Such model will be useful to improve software processes and to better understand the software evolution.

References

[1] Agrawal R. and Srikant R., "Fast Algorithms for Mining Association Rules," *in Proceedings of the 20th International Conference of Very Large Data Bases*, Santiago, pp. 487-499, 1994.

[2] Agrawal R., Imieliński T., and Swami A., "Mining Association Rules Between Sets of Items in Large Databases," *in Proceedings of the ACM SIGMOD International Conference on Management of Data*, Washington, pp. 207-216, 1993.

[3] Ammor O., Lachkar A., Slaoui K., and Rais N., "Optimal Fuzzy Clustering in Overlapping Clusters," *The International Arab Journal of Information Technology*, vol. 5, no. 4, pp. 146-152, 2008.

[4] Antonellis, P., Antoniou D., Kanellopoulos Y., Makris C., Theodoridis E., Tjortjis C., and Tsirakis N., "Clustering for Monitoring Software Systems Maintainability Evolution," *Electronic Notes in Theoretical Computer Science*, vol. 233, pp. 43-57, 2009.

[5] Arisholm E., Briand L., and Foyen A., "Dynamic Coupling Measurement for Object-Oriented Software," *IEEE Transactions on Software Engineering*, vol. 30, no. 8, pp. 491-506, 2004.

[6] Arisholm E., "Empirical Assessment of the Impact of Structural Properties on the Changeability of Object Oriented Software," *Information and Software Technology*, vol. 48, no. 11, pp. 1046-1055, 2006.

[7] Benestad H., Anda B., and Arisholm E., "Understanding Cost Drivers of Software Evolution: A Quantitative and Qualitative Investigation of Change Effort in Two Evolving Software Systems," *Empirical Software Eng*ineering, vol. 15, no. 2, pp. 166-203, 2010.

[8] Bhatt P., Shroff G., and Mishra A., "Dynamics of Software Maintenance," *ACM SIGSOFT Software Engineering Notes*, vol. 29, no. 4, pp. 1-5, 2004.

[9] Briand L. and Wust J., "Empirical Studies of Quality Models in Object-Oriented Systems," *Advances in Computers*, vol. 56, no. 1, pp. 97-166, 2002.

[10] Briand L., Daly J., and Wust J., "A Unified Framework for Coupling Measurement in Object-Oriented Systems," *IEEE Transactions on Software Engineering*, vol. 25, no. 1, pp. 91-121, 1999.

[11] Briand L., Labiche Y., and Sullivan L., "Impact Analysis and Change Management of UML Models," *in Proceedings of International Conference on Software Maintenance*, Amsterdam, pp. 256-265, 2003.

[12] Chidamber S., Darcy D., and Kemerer C., "Managerial Use of Metrics for Object-Oriented Software," *IEEE Transactions on Software Engineering*, vol. 24, no. 8, pp. 629-639, 1998.

[13] Cosine Similarity, https://en.wikipedia.org/wiki/Cosine_similarity, Last Visited, 2016.

[14] Fayyad U., Piatetsky-Shapiro G., and Smyth P., "From Data Mining to Knowledge Discovery in Databases," *American Association for Artificial Intelligence Magazine*, vol. 17, no. 3, pp. 37-54, 1996.

[15] Fowler M., Beck K., Brant J., Opdyke W., and Roberts D., Refactoring: Improving the Design of Existing Code, Boston: Addison-Wesley, 1999.

[16] Ghosheh E., Qaddour J., Kuofie M., and Black S., "A Comparative Analysis of Maintainability Approaches for Web Applications," *in Proceedings of the IEEE International Conference on Computer Systems and Applications*, Dubai, pp. 1155-1158, 2006.

[17] Gui G. and Scott P., "Ranking Reusability of Software Components Using Coupling Metrics," *Journal of Systems and Software*, vol. 80, no. 9, pp. 1450-1459, 2007.

[18] Gupta V. and Chhabra J., "Package Coupling Measurement in Object-Oriented Software," *Journal of Computer Science and Technology*, vol. 24, no. 2, pp. 273-283, 2009.

[19] Jabangwe R., Borstler J., Smite D., and Wohlin C., "Empirical Evidence on the Link between Object-Oriented Measures and External Quality Attributes: A Systematic Literature Review," *Empirical Software Engineering*, vol. 20, no. 3, pp. 640-693, 2015.

[20] Li W. and Henry S., "Object-Oriented Metrics that Predict Maintainability," *Journal of Systems and Software*, vol. 23, no. 2, pp. 111-122, 1993.

[21] Luo C., Li Y., and Chung S., "Text Document Clustering Based on Neighbors," *Data and Knowledge Engineering*, vol. 68, no. 11, pp. 1271-1288, 2009.

[22] Li B., Sun X., Leung H., and Zhang S., "A Survey of Code-Based Change Impact Analysis Techniques," *Software Testing, Verification and Reliability*, vol. 23, no. 8, pp. 613-646, 2013.

[23] MacCormack A., Rusnak J., and Baldwin C., "Exploring the Structure of Complex Software Designs: an Empirical Study of Open Source and Proprietary Code," *Journal of Management Science*, vol. 52, no. 7, pp. 1015-1030, 2006.

[24] Parashar A. and Chhabra J., "An Approach for Clustering Class Coupling Metrics To Mine Object Oriented Software Components," *The International Arab Journal of Information Technology*, vol. 13, no. 3, pp. 239-248, 2016.

[25] Poshyvanyk D., Marcus A., Ferenc R., and Gyimothy T., "Using Information Retrieval Based Coupling Measures for Impact Analysis," *Empirical Software Engineering*, vol. 14, no. 1, pp. 5-32, 2009.

[26] Praditwong K., Harman M., and Yao X., "Software Module Clustering as A Multi-Objective Search Problem," *IEEE Transactions on Software Engineering*, vol. 37, no. 2, pp. 264-282, 2011.

[27] Romanowski C., Nagi R., and Sudi M., "Data Mining In an Engineering Design Environment: or Applications from Graph Matching," *Computers and Operations Research*, vol. 33, no. 11, pp. 3150-3160, 2006.

[28] Sun X., Leungc H., Li B., and Li B., "Change Impact Analysis and Changeability Assessment for A Change Proposal: An Empirical Study," *The Journal of Systems and Software*, vol. 96, pp. 51-60, 2014.

[29] Vanhilst M., Garg P., and Lo C., "Repository Mining and Six Sigma for Process Improvement," *in Proceedings of International Workshop on Mining Software Repositories*, St. Louis, pp. 1-4, 2005.

[30] Vanya A., Premraj R., and Vliet H., "Resolving Unwanted Couplings through Interactive Exploration of Co-Evolving Software Entities-an Experience Report," *Information and Software Technology*, vol. 54, no. 4, pp. 347-359, 2012.

[31] Xie T., Acharya M., Thummalapenta S., and Taneja K., "Improving Software Reliability and Productivity via Mining Program Source Code," *in Proceeding of IEEE International Symposium on Parallel and Distributed Processing*, Miami, pp. 1-5, 2008.

[32] Zaidman A., Bois B., and Demeyer S., "How Webmining and Coupling Metrics Improve Early Program Comprehension," *in Proceedings of the IEEE International Conference on Program Comprehension*, Athens, pp. 74-78, 2006.

[33] Zhong S., Khoshgoftaa T., and Seliya N., "Analyzing Software Measurement Data with Clustering Techniques," *IEEE Intelligent Systems*, vol. 9, no. 2, pp. 20-27, 2004.

Anshu Parashar pursuing his Ph.D. degree from Department of Computer Engineering, National Institute of Technology, Kurukshetra, INDIA. He did B.Techn (2002) and M.Tech. (2008) in Computer Science and Engineering. He has published more than 28 papers in various International, National Conferences and Journals. He has more than 12 years of teaching experience. His area of interest includes software engineering, data mining and object-oriented systems.

Jitender Chhabra working as Professor, Department of Computer Engineering, National Institute of Technology, Kurukshetra, INDIA. He did both his B.Tech and M.Tech. in Computer Engineering from Regional Engineering College Kurukshetra (now National Institute of Technology) as Gold Medalist. He did his PhD in Software Metrics from Delhi. He has published more than 95 papers in various International and National Conferences & Journals including journals of IEEE, ACM, Springer & Elsevier. He has more than 23 years of teaching & research experience. He is author of three books from McGraw Hill including the one Schaum Series International book titled "Programming with C". He is Reviewer of IEEE Transactions, Elsevier, Springer, Wiley & many other Journals. He has worked in collaboration with multinational IT companies HP and TCS in the area of Software Engineering. His area of interest includes software engineering, data mining, soft computing and object-oriented systems.