A New Characterization Scheme of Reusable Software Components

Parvinder Singh Sandhu[1], Hardeep Singh[2], Baljit Saini[3]

[1]Assistant Professor, Department of Computer Science & Engineering, Guru Nanak Dev Engineering College, Ludhiana (Punjab), India. Email:

[2]Professor & Head, Department of Computer Science & Engineering, Guru Nanak Dev University, Amritsar (Punjab), India.
[3]Department of Computer Science & Engineering, Guru Nanak Dev Engineering College, Ludhiana (Punjab), India.

Summary
The software reuse has been gathering the attention of the software industry due to its potential to revamp the software development process. The systematic use of the software reuse is practical and the industrial user data shows that it improves the productivity and quality of the software. However there are issues which have been limiting the wide spread use of software reuse. These relate to software component representation, its storage and retrieval. Understanding and codifying the characteristics of components is essential to the effective management and development of component-based software systems. This paper presents a new characterization scheme for reusable software component based on information retrieval theory. Different organizations of the extracted keywords that represent the semantic feature of the software component are evaluated. This approach allows using uncontrolled vocabulary and automatic indexing of software components that are stored a reusable component library. It could be beneficial for improving the productivity of reuse repository manager by easy identification and retrieval of desired software components.

***Key words**:*
Characterization, Semantic Relation, Repository, Indexing, Importance factor, Relevance factor, Similarity function.

1. Introduction

Software professionals have recognized reuse as a powerful means of potentially overcoming the problem [1] of software crisis [2]-[5] and it promises significant improvements in software productivity and quality [6]. Though significant progress has been made on software reuse, however, there are issues which have been limiting the wide spread use of software reuse process. These issues relate to software component characterization, its storage and retrieval need greater attention of the research community [7]-[8].

In literature, systematic software reuse has been discussed under two broad headings – compositional reuse and generative reuse [9]-[10]. The compositional reuse demands the creation of a Software Reuse Library (SRL), which stores Software Components (SC). Software developer retrieves the prospective SCs from the SRL. He selects the best one satisfying his needs, adapts it, and integrates it into the new application under development. On the other hand, the generative reuse encompasses the reuse knowledge into a tool (such as an application generator) or a language. Then the tool adapts the SC automatically into the new application [12]. The work reported in this paper falls under compositional reuse. For Compositional reuse, several representation methods have been reported in the literature. Frakes and Pole [13] have classified these methods into four categories – Information/Library Science-based methods, AI-based methods, Formal Specification-based methods, and Hypertextbased approaches. Although these representation methods are useful for classification and retrieval of SCs but they have significant limitations as well. Frakes and Isoda [14] define a representation as a language (textual, graphical or other) used to describe a set of objects. Indexing, or classification, is the process used to create a representation using traditional library methods. For Indexing, different indexing techniques can be used. These techniques are classified into two broad categories: controlled vocabulary based - such as enumerated, faceted, keyword based etc. and uncontrolled vocabulary or free-text indexing. In the former, the terms used to describe a component are selected from a predefined set. In the latter, no restrictions are placed on the vocabulary for describing the component.

Enumerated system [15] is a controlled vocabulary method in which every possible class of the vocabulary is predefined. Classes are mutually exclusive and arranged in a hierarchy. Its hierarchical nature makes the system relatively simpler and it is easy to retrieve components. However, in this system, without a thorough knowledge of the subject domain, it is difficult to enumerate all the classes in advance. Secondly, the class hierarchy established is fixed and it is difficult to restructure it i.e. it does not allow SRL designer to add additional levels to the hierarchy.

Another influential technique of controlled-vocabulary type in retrieving SCs is the faceted classification method

Manuscript received August 5, 2007
Manuscript revised August 20, 2007

[16]. In this scheme, vocabularies are indexed into facets. It allows creating a new facet as and when needed and, thus, it overcomes the limitation of fixed hierarchy in the enumerated approach. Many others [17]-[18] have also adapted similar approaches in their works. But these techniques have their own limitations such as manual indexing, difficulties in query formulation and expensive domain analysis. On the other hand, the free-text indexing methods [19]-[20] do not tend to place the components into classes or facets. They are, therefore, not very informative about relationships between the terms used to describe the components. One such system is CATALOG [19], which uses a free text indexing approach for 'C' components. In this system, index terms are automatically extracted from the descriptive headers of 'C' modules and functions. Another similar system, RSL (Reusable Software Library) [20], automatically scans source code files and extracts specially labeled comment statements with attributes such as keywords describing the functionality of components, author, date created, etc. These attributes provide a list of free-text single term indices. The main limitations of these methods are – their domain specific nature, applicable to code components only, less effectiveness due to single keyword based indexing. Knowledge-based methods [21]-[24] include semantic nets and rule-based systems. Semantic nets consist of a directed graph where the nodes represent the SCs and the arcs represent relationships among them. Rule-based methods [21] consist of a set of predefined rules, which are consulted by the system when assisting users in searching components. One important aspect of knowledge-based representations is that they offer powerful ways to express relations between components. On the other hand, the knowledge acquisition problem sometimes proves to be very difficult.

Hypertext-based methods [25]-[26] are similar to semantic nets in that they provide links between the resources. In comparison to knowledge-based system, the user is less aware of these relationships. The major disadvantage of hypertext-based systems is that their construction is considerably more difficult than the other indexing methods.

In this paper, we proposed a new characterization scheme based on information retrieval theory and the concept of lexically related doublet and triplet of words discussed in this paper, which addresses the limitations of earlier models discussed above. The rest of the paper is divided into the following sections. Section 2 describes the proposed representation scheme; section 3 explains the criteria of evaluation of the proposed approach; section 4 describes the implementation and results and finally section 5 gives the concluding remarks.

2. Proposed Scheme

The reusable Software Component (SC) is any part of the software development life cycle that denotes a single abstraction and can be reused. It can be a specification, data model, design, diagram, documentation, function, class, package, module, subsystem, framework etc. Different SCs use different formalisms such as a design component which may be specified by using unified modeling language (UML), similarly, a code component can be a SC which may be written in some programming language, say C++. Despite these differences, there is some commonality among all kinds of components. That commonality is a natural language (say English) documentation associated with every SC, for instance, the function of an SC described in textual form, comments inserted into the source code, naming conventions used in the source code etc. The proposed scheme exploits this natural language description of the functionality of the SC in from of comments and the identifiers used in the SC's design or code. The next section describes the different sources of representation information for an SC.

2.1Sources of Representation Information (RI)

The Representation Information (RI) is the information, from where index terms can be extracted. The following two sources of RI are considered: i) the comments in the source code describing the function of the chunks of code, ii) good naming conventions used in the high-level language source code. Interfaces of the code components (such as function, class, module etc) make high use of verbs, nouns, adjectives and adverbs. These verb phrases convey the functional characteristics of the component. Therefore, function name, constant names and variable-names used make a good source of RI. We exclude reserved because they have no relation with software features. For instance, the pairs 'Close Document', 'End Cursor' etc in OLE components (here, 'Close' & 'End' are function names and 'Document' & 'Cursor' are their arguments respectively) convey functional information about the component.

2.2 Indexing Unit – A Semantic Relation (SR)

An indexing unit (IU) is an atomic unit of an index to be created for an SC. The IU in the proposed scheme consists of a doublet and triplet of words instead of a single word as used in the earlier approaches. The authors call this grouping of words a semantic relation (SR) as they are semantically related. In linguistics, two words are semantically related if they are involved in modifier-modified relationship. Consider the following examples of SRs - count word, close document, open file, copy file, and move file. The words involved in these pairs are semantically related and provide some functional information about the component. In contrast to a single-word IU, an SR based IU provides some additional contextual information about the component because it contains relation between words. Thus, a doublet or triplet of words can form a better indexing unit as compared to a single keyword based indexing unit.

2.3Extraction of SRs from RI

The RI consisting of textual lines are extracted from any of the three sources discussed in section II.2. Each line is either an English sentence (for the sources of type i) or a collection of words extracted from interfaces, design or code of the code components (for the source of type ii). The words in the lines can be divided into two categories – open-class words and closed-class words. Words, which are nouns, verbs, adjectives or adverbs, are called open-class words and are supposed to convey desired functional information about the component. The closed-class words include articles, pronouns, prepositions, conjunctions, interjections, helping verbs and do not convey any functional information. The closed-class words are eliminated from the lines and SRs are extracted from the remaining open-class words. Further, open-class words are stemmed in order to reduce different versions of a word to one form. For instance, the words 'compute', 'computing' and 'computation' will be reduced to 'compute'. Let us illustrate the concept through an example. Consider a line, which after elimination of closed-class words, consists of four open-class words say w_1, w_2, w_3, and w_4. From this line, the doublet of words (SRs) extracted would be: (w_1, w_2), (w_1,w_3), (w_1, w_4), (w_2, w_3), (w_2, w_4), (w_3, w_4) and Triplet of words (SRs) extracted would be (w_1, w_2, w_3), (w_1, w_2, w_4), (w_1, w_3, w_4) and (w_2, w_3, w_4). Let $L_i(w_1, w_2)$ represents i^{th} SR, where w_1, w_2 indicate the first and the second word of an SR and $L_i(w_1, w_2, w_3)$ represents i^{th} SR, where w_1, w_2 ,w_3 indicate the first, second and third word of an SR. The words involved in an SR are stemmed and sorted i.e. $L_I(w_1, w_2)$ = L_i (Sort(Stem(w_1), Stem(w_2)) (similarly for Li(w1, w2, w3)). Where Stem and Sort represent functions for stemming and sorting words of an SR respectively. The algorithm employs the table look-up strategy to eliminate closed-class words and uses Porter algorithm [27] for stemming of words.

2.4 Mathematical formulation of the Model

Full-length For the storage and retrieval of SCs in Software Reuse Library (SRL), SCs are to be indexed. The reuser retrieves SCs by specifying the functionalities of the desired SC. Therefore, the index of the SC should be based on its functional information. Since, the RI describes the functionality of an SC, therefore, the index of an SC is extracted from the RI and this index represents an SC in the SRL. We tried to formulate a model using this representation for retrieving SCs according to their functionalities. The following sub-sections describe the mathematical formulation of the model for indexing and retrieving an SC from SRL.

The proposed model is based on the inverse document frequency of Information Retrieval (IR) theory and concept of SRs discussed in section II.2. The inverse document frequency (idf) defined in the Vector Model of IR theory [28] which is reproduced below.

Let N be the total number of documents in the system and n_i be the number of documents in which an index term k_i appears. Then inverse document frequency idf_i for *ki* is defined in (1).

$$idf_i = \log\left(\frac{N}{n_i}\right) \tag{1}$$

In the Vector Model, the *idf* factor plays an important role. According to this, a low *idf* factor implies that the terms, which appear in many documents, are not very useful for distinguishing a relevant document from a non-relevant document.

Using the concept of *idf* factor in Vector model, we can define a Importance Factor $\eta(w)$ of a word 'w' in RI. The Importance factor of a word in an RI will represent the ability of the word to distinguish a relevant SC from a non-relevant SC. A word appearing very frequently in an RI of an SC is very common word and is likely to appear in all RIs in SRL. Such words are not useful for distinguishing a relevant SC from a non-relevant SC. Let N be the total number of open-class words in an RI and f_W is the frequency of an open-class word w in the RI, then the Importance factor of an open-class word w in the RI can be defined as shown in (2).

$$\eta(w) = \log\left(\frac{N}{f_w}\right) \tag{2}$$

Like idf factor, the value of $\eta(w)$ is lesser for the words with high frequency of occurrence and is higher for words with low frequency of occurrence. On the lines similar to Importance factor of a word, Importance factor of an SR in the RI can also be defined. This will represent the ability of an SR to distinguish a relevant SC from a non-relevant SC. If the words involved in an SR are having high frequency of occurrence then such SRs are likely to appear in the Ris of all SCs in the SRL and hence are not useful for distinguishing a relevant SC from a non-relevant SC. Using the concepts of idf factor and Importance factor of a word, Importance factor of an SR in the RI (means ($\eta(w_1$,

w_2)) for doublet and means ($\eta(w_1, w_2, w_3)$) for triplet of words) can be defined as shown in (3) and (4).

$$\eta(w_1, w_2) = \log\left(\frac{N^2}{f_{w_1} f_{w_2}}\right) \tag{3}$$

$$\eta(w_1, w_2, w_3) = \log\left(\frac{N^3}{f_{w_1} f_{w_2} f_{w_3}}\right) \tag{4}$$

Where w_1, w_2 and w_3 are the words involved in an SR and N is total number of words in an RI. The f_{W_1}, f_{W_2} and f_{W_3} are the frequencies of occurrence of first, second and third words involved in the SR respectively. As it is obvious from (3) and (4) that the value of $\eta(w_1, w_2)$ or $\eta(w_1, w_2, w_3)$ is lesser, if the frequencies of occurrence of the words involved in an SR are high.

Since all the SRs extracted from an RI are not equally significant for using in the representation of an SC. Therefore, to determine the relevance of an SR in the representation of an SC, a *Relevance FactorI* of an SR is defined. The more frequently an SR appears in an RI, the more significant it is, because it has the more functional characteristics of an SC. The relevance factor of an SR, therefore, depends upon the frequency of occurrence of an SR and the significant factor of the SR. It can be defined as the product of the frequency of an SR and Importance factor of an SR as shown in (5) and (6).

$$R = f_{w_1w_2}\,\eta(w_1, w_2) \tag{5}$$

$$R = f_{w_1w_2w_3}\,\eta(w_1, w_2, w_3) \tag{6}$$

Where $f_{w_1w_2}$ and $f_{w_1w_2w_3}$ indicates the frequency of an SR in an RI. Let L_i be the list of SRs extracted from the RI of an SC and R_i (computed by using (5) and (6)) is the corresponding *relevance factor* of L_i for a particular SC. The value of R_i can be normalized as shown in (7).

$$R_i = \frac{R_i - \overline{R}}{\sigma} \tag{7}$$

Where $\overline{R}$ denotes the average and σ denotes the standard deviation of R_i values. The number of SRs extracted from RI would be very large. If all the SRs were used for indexing, then the index size would become very large. To reduce the number of SRs to be used for indexing an SC, an average of R_i as used as cut off value. The most relevant SRs (i.e. SRs having values of R_i greater than cutoff value) would be used in the index.

2.5 Representation of a Software Component

Now, the representation of an SC in SRL consists of the SC's physical contents (say 'C') and its index. Index, further consists of a list of most relevant SRs (L_i) and list of corresponding relevance factors (R_i) Therefore, an SC in the SRL would be represented by a triplet as shown in (8).

$$SC = \{C, L_i, R_i\} \tag{8}$$

Thus SC representation forms the foundation for classifying and retrieving SCs from SRL. The next subsection describes the criteria for finding functional similarity between two SCs.

2.6 Inter-Components Similarity Function

Inter-component similarity function specifies the degree of closeness between the functionalities of two SCs. Common SRs between the representations of two SCs would be used to indicate the functional similarity between two SCs. The more the SRs are common, the more two SCs are similar. To obtain the similarity between two SCs, the Relevance Factors of common SRs between the representations of two SCs are multiplied and summed. This implies that the more relevant SRs get the higher weightage and lesser relevant SRs get less weightage, as the product of large numbers becomes larger and product of small numbers becomes smaller. Therefore, Inter-component Similarity function, S for two components C_P and C_q (pth and qth components of the SCL) can be defined as shown in (9).

$$S(C_p, C_q) = \sum_{i \in L_p \cap L_q} R_{p_i} R_{q_i} \tag{9}$$

Where R_{p_i} and R_{q_i} indicate the lists of relevance factors of common SRs in the representation of components C_P and C_q respectively. L_P and L_q indicate the lists of most relevant SRs in the representation of C_P and C_q respectively. S given by (9) can be used to determine the functional similarities among the components stored in the SRL. It can also be used for retrieving an SC from the SRL by matching the required SC with the SCs stored in the SRL.

3. Evaluation of Proposed Approach

It is tried to evaluate the system in terms of *Precision* and *Recall* criteria. Let S be a set of all software systems contained in a repository. Precision and recall are defined in (10)-(13).

$$Precision = \frac{\sum_{s \varepsilon S} precision_{soft}(s)}{|S|} \tag{10}$$

Where

$$precision_{soft}(s) = \frac{|C_{Actual}(s) \cap C_{Ideal}(s)|}{|C_{Actual}(s)|} \quad (11)$$

And

$$Recall = \frac{\sum_{s \varepsilon S} recall_{soft}(s)}{|S|} \quad (12)$$

Where

$$recall_{soft}(s) = \frac{|C_{Actual}(s) \cap C_{Ideal}(s)|}{|C_{Ideal}(s)|} \quad (13)$$

Where $C_{Actual}(s)$ for a SC say "*s*", is a set of matched or similar SCs of SCL as generated by our software and $C_{Ideal}(s)$ is a set of actually similar SCs, determined manually by the Domain Experts. Using Precision and Recall values we have calculated *F-Value* as a measure of performance evaluation as shown in (14).

$$F\text{–}Value = \frac{2pr}{p+r} \quad (14)$$

Where p is the Precision and r is the Recall of the system.

3. Implementation and Results

There are 43 reusable software components are collected from 'C' based open access repositories. A program is developed in MATLAB 7.2 to test the validity of the proposed characterization scheme and model. SRs are extracted automatically from the input SCs. A table look-up strategy is employed in this algorithm to eliminate closed-class words from the input RI and Porter Stemming algorithm [27] is used to stem the open-class words. This algorithm produces a list of SRs (for both doublet and triplet schemes) along with their frequencies of occurrence and total number of open-class words. Another program implementing computes relevance factors of each SR and retains most relevant SRs. It subsequently computes similarity function (S) between every pair of SCs stored in the SRL contain 43 software components that are categorized as application (e.g. Graphics, spreadsheet based applications), system software (e.g. hardware drivers, complier, linker, loader and other system utilities) and mix category.

It is also tried to represent the SCs with help of a keyword-set containing the Single keyword and their corresponding frequencies. The similarity between the components is calculated by the percentage match of the keywords of two components keeping frequencies as weightage. As the relation between the words is not established, so the performance of the single-word based representation system is far low as compared to the doublet or triplet based schemes.

The new Semantic Relevancy based approach is applied on the stemmed double-word and triple-word schemes and the results are examined in terms of the *Precision*, *Recall*, *F-measure* and *Accuracy* as shown in Table I.

The Relevancy factor based approach has shown the 84.6% *Accuracy* for the Triple word based organization of stemmed words and 0.8947 maximum *F-measure*. In the new approach the results of triple-word scheme are better than the results of double-word; moreover, when compared with the previously used approaches it comes out to be best approach. So, this approach can be recommended for the Domain Relevancy Appraisal.

TABLE I
PERFORMANCE OF NEW CHARACTERIZATION SCHEME OF REUSABLE SOFTWARE COMPONENTS

Algorithm	**Word Scheme**	**Class Type**	**Precision**	**Recall**	**F-Measure**	**Accuracy (%)**
New Semantic-Relevancy Based Approach	**Doublet**	**Class 1**	0.5556	0.5000	0.5263	70.68
		Class 2	0.7000	0.7778	0.7368	
		Class 3	0.7143	0.6667	0.6897	
	Triplet	**Class 1**	0.6923	0.9000	0.7826	84.6
		Class 2	0.8500	0.9444	0.8947	
		Class 3	1.0000	0.6667	0.8000	

4. Conclusion

The new model proposed is based on the concept of SR, doublet and triplet organization of words is used as indexing unit instead of the single word as used by many other models. It overcomes some of the limitations of the earlier models. The doublet and triplet word schemes makes a better indexing unit as it also provides some contextual information as well. This is shown by the similarity function based evaluation of the new model for classification of the Software Components. Also it shows that the use of the model will provide an effective characterization and indexing technique for storing of Software Components in Software Reuse Libraries. Further, this model can be used for automatic indexing of software components and can be helpful in improving the productivity of the reuse repository managers.

References

[1] E. Smith, A. Al-Yasiri, M. Merabti, *A Multi-Tiered Classification Scheme For Component Retrieval*, Euromicro Conference,. 24(2), pp. 882 – 889, 1998.

[2] V. R. Basili , *Software Development: A Paradigm for the Future*, Proceedings of COMPAC '89, Los Alamitos, Calif.: IEEE CS Press, pp. 471-485, 1989.

[3] B. W. Boehm, A Spiral Model of Software Development and Enhancement, *IEEE Computer*, 21(5), pp. 61 – 72, 1988.

[4] M. L. Griss, M. Wosser, Making reuse work at Hewlett-Packard. *IEEE Software*, 12(1), pp. 105 – 107, 1995.

[5] G. Succi, C. Uhrik, M. Ronchetti, Reusability and Portability of Logic Programming, *Journal of Programming Languages Design*, Chapman & Hall, 4(2), pp. 101-114, 1996.

[6] B. Boehm, Managing Software Productivity and Reuse, *IEEE Computer*, 32(9), pp. 111 – 113, 1999.

[7] W. B. Frakes, Kang Kyo, Software Reuse Research: Status and Future, *IEEE Software*, Vol. 31(7), pp. 529-536, 2005.

[8] M. A. Rothenberger, K. J. Dooley, U. R. Kulkarni, N. Nada Strategies for software reuse: A principal component analysis of reuse practices, *IEEE Trans. On Software Engineering*, 29(9), pp. 825-837, 2003.

[9] T. Biggerstaff, C. Richter, Reusability Framework, Assessment, and Directions, *IEEE Software*, 4(2), pp. 41-49, 1987.

[10] C. Kruger, Software Reuse, *ACM Computing Surveys*, 24(2), pp. 131-183, 1992.

[11] W. B. Frakes, S. Isoda, Success Factors of Systematic Reuse, *IEEE Software*, 11(5), pp. 15-19, 1994.

[12] L. S. Levy, A Meta programming Method and Its Economic Justification, *IEEE Trans. On Software Engineering*, 12(2), pp. 272-277, 1986.

[13] W. B. Frakes, T. P. Pole, An Empirical Study of Representation Methods for Reusable Software Components, *IEEE Software Trans. On Software Eng.*, 20(8), pp. 617-630, 1994.

[14] W. B. Frakes, S. Isoda, Success Factors of Systematic Reuse, *IEEE Software*, 11(5), pp. 15-19, 1987.

[15] G. Booch, *Software Components in Ada* (Benjamin/Cummings, Publishing Company, Inc., Redwood City, CA, 1987).

[16] R. Prieto-Diaz, P. Freeman, Classifying Software Reusability, *IEEE Software*, 4(1), pp. 06-16, 1987.

[17] J. Morel, J. Faget, *The REBOOT Environment,* Proceedings of International Workshop on Software Reuse (ISRW-2), E. Guerriery (Ed.), Lucca, Italy, March 24-26, 1993.

[18] E. Damiani, M. G. Fugini, C. Bellettini, A Hierarchy-Aware Approach to Faceted Classification of Object-Oriented Components, *ACM Transaction on Software Engineering and Methodology*, 8(4), pp. 425-472, 1999.

[19] W. Frakes, B. Nejmeh, *An Information System for Software Reuse*, Proceedings of the Tenth Minnor-brook Workshop on Software Reuse, 1987.

[20] B. Burton, R. Aragon, S. Bailey, K. Koehler, L. Mayes, The Reusable Software Library, *IEEE Software*, 4(4), pp. 25-33, 1987.

[21] P. Podgurski, *Behavior Sampling: A Technique for Automated Retrieval of Reusable Components*, Proceedings of 14th International Conference on Software Engineering, pp. 349-360, 1992.

[22] D. Embley, S. Woodfield, *A Knowledge Structure for Reusing Abstract Data Types,* Proceedings of 9th International Conference on Software Engineering, Monterey, pp. 360-368, 1987.

[23] R. Adams, *An Experiment in Software Retrieval*, Proceedings of 4th European Software Engineering Conference (ESEC'93), I. Sommerville and M. Paul (Ed,), LNCS 717, Springer-Verlag, Garmisch-Parten Kirchen, Germany, Sept., 1993.

[24] P. Devanbu, R. Brachman R, P. Selfridge, B. Ballard, LaSSIE: A Knowledge-Based Software Information System, *ACM Comm.*, 34(5), pp. 34-49, 1991.

[25] B. Freitag, *A Hypertext-Based Tool for Large Scale Software Reuse,* Proceedings of the 6th Conference on Advanced Information Systems Engineering (CAISE'94), Utrecht, Netherlands, 6-10 June 1994, pp. 283-296, 1994.

[26] P. Hitchcock, B. Wang, Intersect Hypertext DM, *Journal of Information and Software Technology*, 34(9), pp. 573-592, 1992.

[27] http://snowball.tartarus.org/.

[28] J. K. Spärck, A statistical interpretation of term specificity and its application in retrieval, *Journal of Documentation*, Vol. 60, pp. 493-502, 2004 .

Parvinder S. Sandhu is working as Assistant Professor in the Department of Computer Science and Engineering with Guru Nanak Dev Engineering College, Ludhiana (Punjab). He is Master of Engineering in Software Engineering, M.B.A. and Bachelor in Computer Engineering from NIT, Kurukshetra. He is doing research work leading to Ph.D. with Guru Nanak Dev University, Amritsar. He has published 09 research papers in referred International journals and 12 papers in renowned international conferences. His current research interests are Software Reusability, Software Maintenance and Machine Learning.

Hardeep Singh is working as Professor in the Department of Computer Science and Engineering with Guru Nanak Dev University, Amritsar, India. and the other authors may include biographies at the end of regular papers. His date of birth is Feb. 16 and he has got twenty years of teaching experience. He is member of high profile committees of Government of India related with the technical education. He is Doctorate in Modeling and Design of Software Metrics for Object Oriented Systems. He has thirty five International and National publications to his name. He is live interest in Software Engineering, Object Oriented Paradigm, Management Information & Decision Support Systems, Ergonomics, Computer Networks, Artificial Intelligence.

Baljit Saini is currenly doing her Masters from Guru Nanak Dev Engineering College, Ludhiana and doing research on the characterization of components in the software repositories under supervision of Prof. Parvinder S. Sandhu. She did her Bacholoes in Computer Science and Engineeirng from Punjab technical university Jaandhar.