I.J. Information Engineering and Electronic Business, **2013, 6, 33-39**
Published Online December 2013 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijieeb.2013.06.04

A Meta Level Data Mining Approach to Predict Software Reusability

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

Megha Rathi
Jaypee Institute of Information Technology, Noida, India
E-mail: megha.rathi@jiit.ac.in

***Abstract* —** Software repositories contain wealth of information about software code, designs, execution history, code and design changes, bug database, software release and software evolution. To meet increased pressure of releasing updated or new versions of software systems due to changing requirements of stakeholder, software are rarely built from scratch. Software reusability is a primary attribute of software quality which aims to create new software systems with a likelihood of using existing software components to add, modify or delete functionalities in order to adapt to new requirements imposed by stakeholders. Software reuse using software components or modules provide a vehicle for planning and re-using already built software components efficiently. In this paper, we propose a framework for our approach to predict software reusable components from existing software repository on the basis of (1) stakeholders intention (requirement) match and (2) similarity index count for better reuse prediction. To effectively manage storage and retrieval of relevant information we use concept of situational method engineering to match and analyze the information for reuse. We use Genetic algorithm, Rabin Karp algorithm for feature selection and classification and k-means clustering methods of data mining to refine our results of prediction in order to better manage and produce high quality software systems within estimated time and cost.

***Index Terms* —** Data mining, Software reuse, requirements engineering, situational method engineering, software reusability and prediction.

1. Introduction

Computers have become an integral part of today’s life and this has led to development of sophisticated and complex computer-based systems. This shift has shown tremendous improvements in hardware performance, architectures, storage capacity and other user-friendly options in last five decades. This has revolutionized the computing world as one can now design and use software according to his/her own requirements and can get better execution results in a quick time span. Industrial data show that there has been an exponential growth in the size of software systems for past 40 years [1]. Thus, there is an increased pressure on software engineers to manage and produce high quality software system within estimated time and cost. In order to attain the optimal software, programmers reuse the existing software components or libraries, rather than developing similar code from scratch [2]. Reusable modules and components provide engineers the confidence of increased probability of elimination of bugs with prior testing when a change in implementation is required.

Software reuse is a promising area for achieving software productivity and software quality in this evolutionary era. The main focus of software reusability is on correctness and reliability which are the two aspects of software quality. As a result it also helps in reducing development time and cost.

For any software development Requirement Engineering (RE) is crucial to develop effective software systems by reducing software errors (by translating imprecise, incomplete requirements into complete, precise and formal specifications) at the early stage of the development of software. Requirement engineering (RE) according to [3] is "a sub discipline of systems engineering and software engineering that is concerned with determining the goals, functions, and constraints of hardware and software systems." The changes made in this level of software development life cycle will ripple down the life cycle and will accordingly affect design, coding and implementation. Requirement analysis in RE aims to determine user expectations (for new or modified system) by taking into account possible conflicting requirements of the various stakeholders. This helps in reducing repair cost the chances of project cost overruns.

In recent past software reuse has gained light in research community and has become important in various aspects of software engineering techniques and methods. In the field of RE documentation is important for reuse in order to support accommodation of new or modified features and release correct and reliable software. Domains like data mining, machine learning, neural networks etc. are very useful in generating relevant information for predicting software reusability.

The goal of this research is to present a framework for tool development to predict software reusability using well established techniques of data mining (Genetic algorithm, Rabin Karp, k-means clustering). The requirement engineering phase consists of representation of stakeholder's interest or requirement as intention match. In order to predict reusable components or modules we analyze meta-level information extracted from stakeholder's intention (requirement) in subsequent stages. For storage and retrieval of relevant information we use concept of Situational Method Engineering (SME) [4] which assumes existence of a method repository from where method (s) of interest are retrieved, modified or assembled into a new method that is subsequently stored in repository.

The remainder of this paper is organized as follows: next section presents related work. Section 3 describes our proposed framework and methodology for predicting reusable components followed by a discussion on discussion on matching, retrieval, verification, classification and clustering of requirements and its meta-level information required to predict software reusability with a similarity index count using proposed framework in section 4 and finally section 5 presents the conclusion.

2. Related Work

The concept of reuse was introduced to world in 1968 and academia got attracted to it in late 1970s which led to a new way of designing and developing software systems in less time and cost [5]. McClure[6] defines software reuse as "process of creating software systems and software projects deliverable from predefined or prefabricated components or assets" whereas Yu[7] defines software reuse as "software engineering activities which focus on the identification of reusable software for straight import, reconfiguration, and adaptation for new computing system applications" and Feeler[8] defines software reuse "is an engineering activity that focuses on the recognition of commonalities of systems within and across domains. It consists of the creation of models with different abstractions (ranging from code components to domain models) and their use during the engineering of an application. Thus, the focus is on the growth and utilization of technology base."[9] presents an approach which divides reuse activity into six steps performed at each phase in preparation for the next phase. These steps are: (a) problem analysis and identifying available solutions for developing a reuse strategy (b) identifying a solution structure for the problem following the reuse plan (c) reconfiguring solution structure to improve reuse at the next phase (d) acquiring, instantiating, and/or modifying existing reusable components (e) integrating reused and any newly developed components into the products (f) evaluating the products.

Several decades of study have acknowledged reuse as a powerful and potential way of fighting software crisis problem [10, 11, 12, 13, 14, 15, 16, 17]. Other studies like [18, 19, 20, 21, 22, 23, 24] discuss how software reuse can be helpful in improvements in software quality and productivity. Reusability of a component can be measured in two ways-qualitative and empirical. The former approach relies on subjective value attached to guidelines to which software system corresponds. [25] is a manual process whereas latter takes into account static software metrics that directly or indirectly addresses the different attributes of reusability. Metrics and tool presented in literature [26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37] identifies attributes, reusability characteristics based on software quality and productivity to evaluate software reusability. These studies use tools, matrices and perform extermination for their result predictions. Various studies on reusability have conducted in past including [38, 39, 40, 41, 42, 43, 44, 45, 46, 47] and its research directions [48, 49] and on reusability in practice [18, 50, 51, 52]. Developers are exploring and adopting many of these reuse approaches for advanced searching, matching, and modeling tools [53].

3. Basic Terminology and Proposed Framework

We propose a framework that classifies the user intention (requirement) and analyses data collected as a result of classification for calculating similarity index between already existing software components and required components expresses as intention by user for efficient reuse prediction. Intention here refers to the requirement for which he/she wants to search software repository for available software components for reuse. This data can then be used by software engineer for building software system matching user intention within estimated cost and time. The whole idea is presented in Fig. 1 below.

The proposed framework consists of five main modules namely, stake holder's intention, classifier, analyzer, clusteror and categorizer. Out of these five modules three marked with dark boundaries namely, classifier, analyzer and clusteror are further divided into sub modules. These inner sub modules are explained in Fig. 2 (a) (b) and (c) respectively. Next section discusses the working of proposed framework and steps taken for predicting software reusability. The steps are as follows:

Step 1: The user interacts with graphical user interface to upload his/her intention. For this purpose we plan to provide a web based interface for interaction with the system. The classifier reads requirement stated by user and first performs feature selection. We use Genetic Algorithm (GA) [54] as feature selection method to remove the irrelevant and redundant information (features) from the data (inputted as stakeholder's requirement) to improve the performance of proposed model for analyzing reusability. With the help of GA we can identify and separate important keywords from a sentence or phrase entered by user. Working of GA is a four step process (a) attributes are selected (b) a fitness function is computed (c) fitness is evaluated and (d) reproduction. Once the keywords are separated pattern matching is done in order to match the keywords with existing keywords in repository.

We use Rabin Karp Algorithm [55] for this purpose. It is an efficient string matching algorithm which can readily search for instances of sentences from the given source material, ignoring details such as case and punctuation. A practical application of Rabin Karp is detecting plagiarism. The results produced by this method more accurate than other existing methods. This classification is made on the basis of keywords already stored in the database. Here, we are using the concept of situational method engineering [4] which assumes existence of a method repository from where method(s) of interest are retrieved, modified or assembled into a new method that is subsequently stored in repository. The user can express his/her requirement in the form of a sentence or phrase. This pattern matching classifier will help in minimizing the requirement only to important words and not phrases like “the”, “is”, “wants”, “I”, “we”, “use” etc. The classification results produced are stored in database-1 as shown in Fig. 1 below.

Step 2: In the next step analyzer processes data obtained from classification stored in database-1. Analyzer here works as a two stage process. In the first step it searches relevant keywords obtained as input from step 1 into software repository using various permutation and combinations which will be helpful in searching appropriate module for reuse.

In other words, it considers synonyms for searching. The list of possible synonyms is maintained in central software repository. The information sharing between proposed framework and central repository is depicted repository. Fig. 3 explains abstract view of central in Fig. 4. In the second step it performs matching between proposed requirement and existing software modules. We are taking into consideration the criteria for minimum matching to take place as 40% which we are calling as passing criteria. The analyzer stores this matched information along with percentage of passing criteria in the form of table stored in database-2. The example table is presented in Table-1 below:

Step 3: Next step is to perform clustering on the basis of percentage match obtained from analyzer. We use k-means [56] to group the similar data into clusters the advantage of using k-means is that, it is faster with a large number of variables and produces tighter clusters. In this phase all those modules which are similar in nature will be clubbed together. After clustering of similar modules similarity index count (SIC) is computed using [57]. Higher value of SIC indicates more similarity with the expresses requirement and lower value indicates less similarity. The modules with low SIC value are not rejected rather they are also recommended after recommending modules with higher SIC count to software engineer to find and choose the correct matched modules. This is done so that if some functionality is missing from higher SIC indexed modules than the same can be looked in lower indexed value modules. This will help in maximum reusability. This information is stored in the form of table in database-3. The sample view of this table is presented in Table-2 below. Finally software engineer is going to read Table-2 and will decide and pick those components which match to the user’s intention the most. Thus, using this approach software engineer can decide from the pool of data what to choose based on stake holder’s intention match and similarity index count.

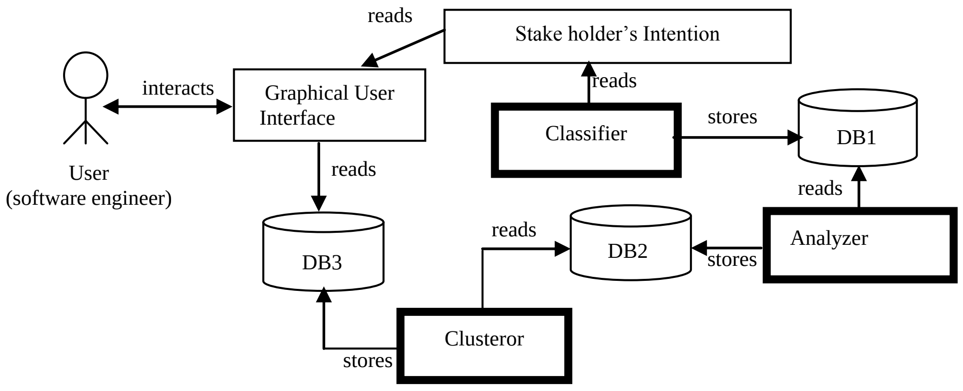

Figure 1. Framework for software component reusability model

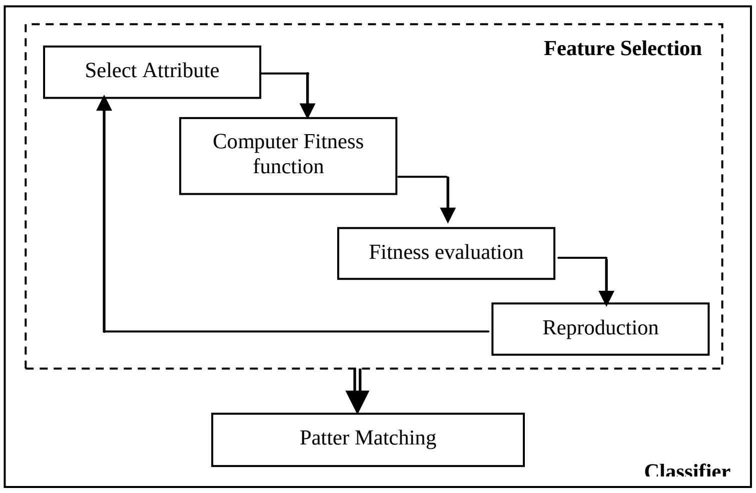

Figure 2(a). Detailed view of Classifier

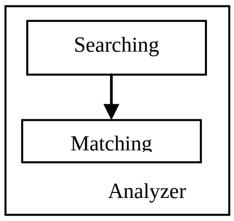

Figure 2(b). Detailed view of Analyzer

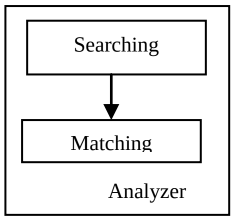

Figure 2(c). Detailed view of Clusteror

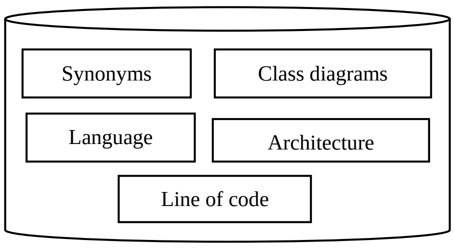

Figure 3. Abstract view of central software repository

4. Tool Sketch for Proposed Approach

We considered the issue of providing tool support for our proposed approach. The nature of this support is sketched in Fig. 4. As illustrated there are three small databases and one central software repository. These databases are connected with each other for efficient retrieval and storage of required information. Database-3 interacts with central repository for searching possible match modules along with percentage of match and with database-1 to know the requirement for searching and matching. Database-2 interacts with database-3 for clustering and calculating similarity index values for processed information stored in database-3 and with editor to display the results for effective analysis of reusability of software components. The results of the query are shown in the form of table (Table-2 depicts the sample view of same). Database-1 on the other hand interacts with query handler to fetch the stake holder's intention for processing.

TABLE 1. SAMPLE VIEW OF RESULTS OBTAINED FROM ANALYZER

Intention to match	Possible matches	Percentage of match
Portfolio management	Investment management	80%
	Portfolio manager	55%
	….	…

TABLE 2. SAMPLE VIEW OF RESULT COMPUTATION FOR CLUSTEROR

Clusters	Module Numbers	Similarity Index Count
Cluster 1	M1, M4, M5	80%
Cluster 2	M2, M3	74%
…..	…	…
Cluster n	Mn, Mn+2	@%

After analysis, the user can choose the components he wishes to use and can update the central repository with new software module any time after finishing the task through editor interface. The reason for storing this information is to use it for future use (re-use). The software engineer can at any time access this information for further analysis.

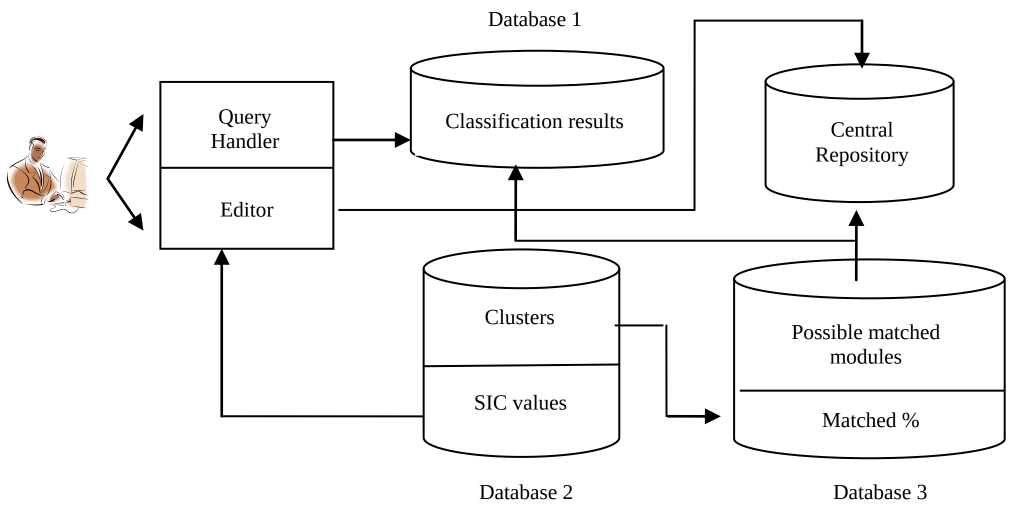

Figure 4. Tool sketch for proposed Approach

5. Conclusion

This matching and analysis capability to compute similarity (with index count) within software modules and components can be helpful to reduce developer's effort. This approach can be helpful in identifying those parts of the software that can be further reused thereby significantly reducing effort, cost and time. We are currently in process of implementing the proposed tool. So far, web enabled interface, classifier and analyzer has been completed and we are now looking to implement clusteror.

References

[1] Humphrey W S. The Future of Software Engineering: I [R]. Watts New Column, News at SEI, 2001, 4(1).
[2] Krishna T M., Vasumathi D. A Study of Mining Software Engineering Data and Software Testing [J]. Emerging Trends in Computing and Information Sciences, 2011, 2(11): 598-603.
[3] Laplante P A. What Every Engineer Should Know about Software Engineering [M]. Taylor & Francis Group, Boca Rotan, FL, 2007.
[4] Harmsen, F., Brinkkemper, S., Han, J L O. Situational Method Engineering for Information System Project Approaches [C]. In: Proceedings of the IFIP WG8.1 Working Conference on Methods and Associated Tools for the Information Systems Life Cycle (IFIP '94), September 1994, 169-194.
[5] Gomes P., Bento C. A Case Similarity Metric for Software Reuse And Design [J]. Artificial Intelligence for Engineering Design, Analysis and Manufacturing, 2001, 15(1.1): 21-35.
[6] McClure C. Software Reuse Techniques [M]. Prentic-Hall, Inc., 1997.
[7] Yu D. A view on Three R's (3Rs): Reuse, Re-engineering, and Reverse Engineering [C]. Software Engineering Notes, July 1991, 16(3):69.
[8] Feiler P H. Reengineering: an engineering problem [R]. Software Engineering Institute, Carnegie Mellon University: Special Report, 1993.
[9] Kyo K C., Sholom C., Robert H., James P., Spencer P A. A Reuse-Based Software Development Methodology [R]. Application of Revisable Software Components Project Special Report, 1992.
[10] Smith E., AI-Yasiri A., Merabti M. A Multi-Tiered Classification Scheme for Component Retrieval [C]. In: Proceedings 24th Euromicro Conference, August 1998, 2:882 -889.
[11] Basili V R. Software Development: A Paradigm for the Future [C]. In: Proceedings of Computer Software and Applications Conference (COMPAC'89), September 1989, 471-485.
[12] Basili V R., Rombach H. D. The TAME Project: Towards Improvement-Oriented Software Environments [J]. IEEE Transaction on Software Engineering, 1988, 14(6): 758-771.
[13] Boehm B W., Ross R. Theory-W Software Project Management: Principles and Examples [J]. IEEE Transaction on Software Engineering, 1989, 15(7): 902-916.
[14] Boehm B W. A Spiral Model of Software Development and Enhancement [J] Computer, 1988, 21 (5):61 -72.

[15] Griss M L. Software Reuse: From Library to Factory [J]. IBM Systems Journal, 1993, 32(4):548-566.
[16] Poulin, Yglesias K. Experiences with a faceted Classification Scheme in a Large Reusable Software Library (RSL) [C]. In: Proceedings 7th Annual International Computer Software and Applications Conference (COMPSAC '93), 1993, 90-99.
[17] Succi G., Benedicenti L, Vernazza T. Analysis of the Effects of Software Reuse on Customer Satisfaction in an RPG Environment [J]. IEEE Transaction on Software Engineering, 2001, 27(5): 473-479.
[18] Boehm, B. Managing Software Productivity and Reuse [J]. Computer, 1999, 32(9):111 - 113.
[19] Frakes W B., Fox C J. Quality Improvement Using a Software Reuse Failure Modes Model [J]. IEEE Transaction on Software Engineering, 1996, 22(4): 274-279.
[20] Griss M L., Wosser M. Making reuse work at Hewlett-Packard [J]. IEEE Software, 1995, 12(I):105 - 107.
[21] Lim W. Effects of Reuse on Quality, Productivity, and Economics [J]. IEEE Software, 1994, II (5): 23-30.
[22] Meyer B. The Reusability Challenge [J]. IEEE Computer, 1996, 29(2): 76 -78.
[23] Mili H., Mili F., Mil A. Reusing software: issues and research directions [J]. IEEE Transaction on Software Engineering, 1995, 21(6):528 - 562.
[24] Gill N S. Importance of Software Component Characterization for Better Software Reusability [C].In: Proceeding of ACM SIGSOFT Software Engineering Notes, 2006, 31(I):1-3.
[25] Jeffrey S. Poulin. Measuring Software Reusability [C]. In: Proceedings of the Third International Conference on Software Reuse, November 1994.
[26] Prieto-Diaz, Ruben, Freeman P. Classifying software for Reusability [J]. IEEE Software, 1987, 4(1):6-16.
[27] Selby, Richard W. Quantitative Studies of Software Reuse [J]. Software Reusability, 1989, II: 213-233.
[28] 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.
[29] Caldiera, Gianluigi, Victor R. Basili. Identifying and Qualifying Reusable Software Components [J]. IEEE Software, 1991, 24(2): 61-70.
[30] Karlsson, Even-Andre, Sindre G., Stalhane T. Techniques for Making More Reusable Components [R]. REBOOT Technical Report, 1992.
[31] 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.
[32] 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, June 1993, 271-274.
[33] Torres, William R., Samadzadeh M H. Software Reuse and Information Theory Based Metrics [C]. In: Proceedings of Symposium on Applied Computing, Kansas City, April 1991, 437-46.
[34] 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, 1991, 18-22.
[35] NATO. Standard for Management of a Reusable Software Component Library [R]. NATO Communications and Information Systems Agency, 1991.
[36] RAPID. RAPID Center Standards for Reusable Software [R]. U.S. Army Information Systems Engineering Command, 1990.
[37] Piper, Joanne C., Wanda L. Barner. The RAPID Center Reusable Components (RSCs) Certification Process [R]. U.S. Army Information Systems Software Development Center.
[38] Morisio M., Ezran M., Tully C. Success and Failure Factors in Software Reuse [J]. IEEE Transaction on Software Engineering, 2002, 28(4): 340-357.
[39] Bockle G, Lements P., McGregor J D., Muthig D., Schmid K. Calculating ROI for Software Product Lines [J]. IEEE Software, 2004, 21(3):23-31.
[40] Deelstra S., Sinnema M., Nijhuis J. Bosch J. COSV AM: A Technique for Assessing Software Variability in Software Product Families [C]. In: Proceedings 20th IEEE International Conference on Software Maintenance, Sept 2004, 458-462.
[41] Bosch J., Juristo N. Designing Software Architectures for Usability [C]. In: Proceedings 25th International Conference on Software Engineering, May 2003, 757-758.
[42] Bosch J. Architecture-Centric Software Engineering [C]. In: Proceedings 24th International Conference on Software Engineering, May 2002, 681-682.
[43] Bosch J. Software Product Lines: Organizational Alternatives [C]. In: Proceedings 23rd International Conference on Software Engineering, May 2001, 91-100.
[44] Bosch J. Design and Use of Industrial Software Architectures [C]. In: Proceedings Conference on Technology of Object-Oriented Languages and Systems, June 1999, 404-404.
[45] Coplien J., Hoffman D., Weiss D. Commonality and Variability in Software Engineering [J]. IEEE Software, 1998, 15(6): 37-45.
[46] Esteva J C. Automatic Identification of Reusable Components [A]. In: Proceedings of 7th International Workshop Computer-Aided Software Engineering, Toronto, 1995, 80-87.
[47] Klein J., Price B., Weiss D. Industrial-Strength Software Product-Line Engineering [C]. In:

Proceedings 25th International Conf. Software Engineering, May 2003, 751-752.

[48] Devanbu P T., Perry D E., Poulin J S. Guest Editors Introduction: Next Generation Software Reuse [J]. IEEE Transaction on Software Engineering, 2000, 26(5):423-424.

[49] Frakes W B., Kang K. Software Reuse Research: Status and Future [J]. IEEE Transaction on Software Engineering, 2005, 31(7): 529 - 536.

[50] Tomer A., Goldin L., Kuflik T., Kimchi E., Schach S R. Evaluating Software Reuse Alternatives: A Model and its Application to an Industrial Case Study [J]. IEEE Transaction on Software Engineering, 2004, 30(9):601-612.

[51] Rothenberger M A., Dooley K J., Kulkarni U R., Nada N. Strategies for Software Reuse: A Principal Component Analysis of Reuse Practices [J]. IEEE Transaction on Software Engineering, 2003, 29(9): 825-837.

[52] Lee N Y., Litecky C R. An Empirical Study of Software Reuse with Special Attention to Ada [J]. IEEE Transaction on Software Engineering, 1997, 23(9): 537-549.

[53] Kazman R, Abowd G., Bass L., Clements P. Scenario-Based Analysis of Software Architecture [J]. IEEE Software, 1996, 13(6): 47-55.

[54] Lanzi P L., P. di Milano. Fast Feature Selection with Genetic Algorithms: A Filter Approach [C]. In: Proceedings of IEEE International Conference on Evolutionary Computation, April 1997, 537-540.

[55] http://en.wikipedia.org/wiki/Rabin–Karp_algorithm.

[56] Kanungo T., Mount D. M., Netanyahu N. S., Piatko C. D., Silverman Wu A. Y. An Efficient k-Means Clustering Algorithm: Analysis and Implementation [J]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2002, 24(7): 881 – 892.

[57] Goldberg M K. Hayvanovych M., Ismail M. M. Measuring Similarity between Sets of Overlapping Clusters [C]. In: Proceedings of the 2010 IEEE Second International Conference on Social Computing (SOCIALCOM '10), 2010, 303-308.

Authors' Profiles

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.

Megha Rathi: She is Assistant Professor (Grade II) at Jaypee Institute of Information Technology, India. She holds a Masters of Technology and a Bachelor of Engineering degree in Computer Science and Engineering. Currently she is pursuing her PhD in Computer Science and Engineering. Her areas of interest are Database systems, Software Engineering, Software Testing and Artificial Intelligence.