International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:1, No:4, 2007

# Software Architecture Recovery

Ghulam Rasool, and Nadim Asif

***Abstract*—**The advent of modern technology shadows its impetus repercussions on successful Legacy systems making them obsolete with time. These systems have evolved the large organizations in major problems in terms of new business requirements, response time, financial depreciation and maintenance. Major difficulty is due to constant system evolution and incomplete, inconsistent and obsolete documents which a legacy system tends to have. The myriad dimensions of these systems can only be explored by incorporating reverse engineering, in this context, is the best method to extract useful artifacts and by exploring these artifacts for reengineering existing legacy systems to meet new requirements of organizations. A case study is conducted on six different type of software systems having source code in different programming languages using the architectural recovery framework.

***Keywords*—**Reverse Engineering, Architecture recovery, Architecture artifacts, Reengineering.

## I. Introduction to Architecture Recovery

THE software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them[1]. Software architecture design is concerned with gross organization and global control structure of a system. Architecture bridges the gap between the requirements and implementation of the system. Software architecture is very important concern due to understanding, analysis, reusability, evolution and management of legacy systems.

We define architecture recovery as a process of identifying and extracting higher level of abstractions from existing software systems [2]. Architecture recovery and reengineering to handle legacy code is critical for large and complex systems. Architecture recovery deals with the issues of recovering the past design decisions that has been taken by the experts during the development of a system [3]. These are decisions that has been lost due to some reasons; not documented, document revisions or developer have left or unknown (i.e. assumptions not initially taken in account). In architecture recovery the research is continue on issues of interoperability: techniques for detecting component mismatch and bridging them [4]. The recovery process can be assisted by different tools available in the market like Dali [5], PBS[6], Imagix4D[7] and Bauhaus[8]. No one tool can perform all the tasks required for architecture recovery. So we used our custom built tool DRT having excellent features.

Ghulam Rasool is with the COMSATS Institute of Information Technology, Lahore. 1-Km, Defense Road, off Raiwand Road, Lahore, Pakistan (corresponding author, phone: 0092-333-4520196; fax: 0092-42-9203100; e-mail: ghrasool@hotmail.com).

Prof. Dr. Nadeem Asif, is with The University of Lahore, 1-Km Raiwand Road, Lahore, Pakistan (e-mail: nasif@ulhr.edu.pk).

## II. Architecture Representation/Properties

An architecture has different stakeholders with different concerns. Architectural representations enable software developers to explicitly describe, access and manage the architecture of software systems. Architecture representation consist of structural and non-structural information about software architecture. Structural information are components and connectors describing the configuration of a system and non structural information are architectural properties[5]. Architectural properties are for example, safety patterns, communications patterns, behavioral patterns ,structural patterns and creational patterns. The recognition of different type of similar patterns is very important knowledge for understanding the existing legacy systems and architecture recovery. The user understand the conceptual and concrete architecture of the system through architectural documents , design patterns , source code and architectural properties. The architecture properties can not be ignored during the recovery of different architecture artifacts.

### *A. Architectural Descriptions*

The language for specifying an architecture should ideally be expressive, well-defined, abstract, concise and compact For example ADL [9] for specifying an architecture recovery results is used that permits formal reasoning and supported by tools. Most ADL are formally defined but their actual use in industry is very limited. It is still interesting to evaluate whether formality is of importance to architecture extraction. A lexical based regular extraction technique is used as a specification language to extract different artifacts from source code of different programming languages. It allow the user to use the specifications according to the requirements based on action and analysis in the regular expressions for task at hand.

### *B. Related Approaches*

There are different approaches for reverse engineering, which can be attempted at different level of abstractions [10] . These approaches are related to our work. The structural recovery techniques are mostly used for components recovery. The Murphy's Reflection model [11] allow the user to test the high level conceptual model of the system against the existing high level relations between the components of the system. The recovery approaches are classified as follows according to type of information they provide:

- Data Flow based approaches [12].

- Knowledge based approaches [13].
- Design patter based approaches [14].
- Program slicing based approaches [15].
- Formal method based approaches [16].
- Program comprehension based approaches [17].
- Domain based approaches [18].
- Clustering based approaches [19].
- Concept analysis approaches [20].
- Machine Learning approaches [21].
- Metrics Based approaches [22].
- Structural Based approaches[23]

We used the unification of best approaches for extraction of different artifacts from the source code and documents. The best features of domain based, program comprehension based, design pattern based and clustering based recovery approaches are used to recover the architecture of software systems under study. Regular expressions are used to write different pattern specifications to extract desired artifact at different levels of abstractions.

## III. Framework for Architecture Recovery

The Proposed Framework integrates the existing architecture recovery tools to support architecture recovery process. In many cases, architectural information is available as block-line diagrams [9]. However, most architecture information is inherent and hidden in different styles and views of source code and design documentation. The extraction of architectural information is required using different techniques and tools.

Fig. 1 sketches an overview of the proposed framework for an architecture recovery. The input of the recovery process is the source code, design documentation, domain knowledge, artifacts recovered from pattern based, clustering techniques and expert knowledge if any experts or rational exists. Finally results are represented in different formats and styles.

The recovery of design documentation and domain knowledge delivers additional information into already existing abstractions such as data flow diagrams and support the generation of additional software views, for example state transition diagrams, component diagrams and architecture descriptions.

Source code and required artifacts can be extracted with the help of reverse engineering tools. Reverse engineering tools perform static analysis on the code and extract information like call graphs, cross-reference tables, data flow diagrams, quality metrics, hierarchies in classes, relationship and other useful information.

Reverse engineering tools provide a higher level of abstraction since information that is not of interest for the specific view is excluded. The results of reverse engineering tools are analyzed and verified with some of the available source of information (documents, source code and comments available in the source code). User knowledge is incorporated in the tool to write different lexical specifications. RE tool generate different views which can be used to recover the architecture of the system. Similarly we can use the bottom up approach and can take artifacts as an input and can generate different software views.

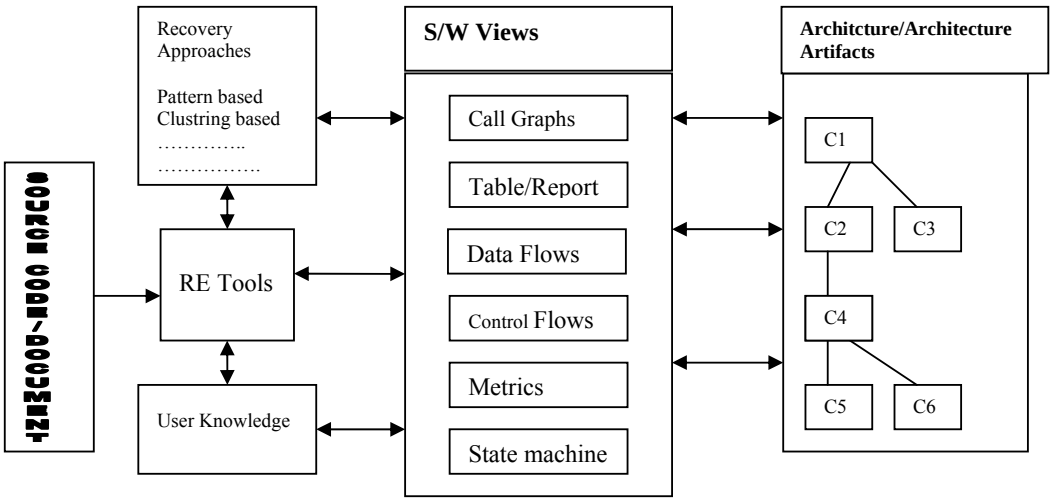


Fig. 1 Framework for architecture recovery

Based on our experience and knowledge we determined the following strategy for architecture recovery:

1. Study the different architecture recovery approaches (such as Domain based, Design patter based, clustering based etc).
2. Develop architecture conceptual model and formed architecture hypothesis regarding the system and its structure.

3. Analyze, verify and refine the architecture hypothesis against the software system under study.

6. Use the reverse engineering tools.
7. Conducted a case study on the code of five different programming languages software.
8. Used existing documents for understanding of system structure and its components.

## IV. Recovery Process

The selection of architecture recovery process is the key concern for extracting the artifacts from the legacy system architecture. Different research groups define the process according to the nature of the system. Recovery Process adopted in our study consists of following Phases [3].

1) Architecture concepts
2) Legacy architecture analysis
3) Extraction
4) Abstraction
5) Evaluation
6) Presentation.

In first step we built the hypothesis about the architecture of the existing system. In second phase we analyze the hypothesis developed in the first phase with the help of tools. The next phase extracts the different artifacts from the system using extraction techniques and Reverse Engineering tools. Abstraction process produces architecture styles and views at different level of abstractions. In evaluation stage, the results are evaluated and compared with existing sources of information. Finally the recovered architecture is represented in different formats, styles, and UML notations.

4. Generate different software views and architecture styles of the examined system.
5. Iterative Use of architecture recovery Process.

## V. Tool Support

The artifacts from the legacy systems can be extracted by using different tools available in the market like Imagix 4D, Rigi [24] and Refine/C[5]. These tools have certain limitations like language dependency and compiled code. Due to these reasons we used custom built DRT [25,26] which supports the limitations mentioned above. The results of extracted artifacts from different programming languages source code are shown in Table I. Our custom-oriented tool supports the following features.

1) It is language independent and used in a study of source code and documents of five different programming languages to extract different required artifacts.
2) It take source code as input which may be incomplete , uncompilied or have errors.
3) User can write specification of similar types to extract artifacts from code and documents of different programming languages software's.
4) Artifacts can be presented in different formats and styles.
5) Internal/External knowledge can be included in the tool to extract the desired artifacts.
6) The matched patterns may be further analyzed to extract further relationships between the patterns and may be represented in different formats.
7) The vocabulary of the tool can be extended according to nature of maintenance task at hand and requirement of the source code.
8) The hierarchal and abstract pattern specifications may be used to extract the required artifacts.
9) It can filter out the false matches by action pattern specifications.

TABLE I
Extracted Artifacts

| Software | Size on disk | Lines of Source code in KLOC | Files | | Include files | Functions | Blank lines | Lines of Comments |
|---|---|---|---|---|---|---|---|---|
| | | | Total Files | Code files | | | | |
| Alligance Game/C++ | 823MB | 450 | 7629 | 1341 | 3463 | 612 | 71964 | 74679 |
| Elm/C++ | 8.05MB | 35 | 479 | 455 | 905 | 422 | 6566 | 7686 |
| Tac_Plus/C | 592KB | 20 | 50 | 50 | 153 | 310 | 3181 | 2600 |
| Mining/Java | 150KB | 6 | 6 | 6 | 11 | 126 | 684 | 1088 |
| Monica/VB | 2.50MB | 18 | 50 | 33 | - | 621 | 5 | 50 |
| Drawing Editor/Pascal | 1.53MB | 8 | 45 | 10 | - | 252 | 847 | 524 |
| Client Messaging/Cobol | 1.40 MB | 20 | 47 | 23 | - | - | 194 | 7000 |

In addition to above artifacts extracted in the Table I, the specifications can be used to extract further artifacts required for architecture recovery. The technique has been used to extract classes, inheritance, Cobol files, Record formats, functions, function calls and the relations between different entities. For example the following specification is used to extract different procedure names from a Source code of Pascal program(Quartz Demo 2.1) as shown in table2.

**Pattern**: (\procedure|Procedure)((\s+\w+\d+)|(.*))

Similarly we can write different specification to extract our required artifacts from source code of different programming language. Expressions allow us to attach actions and analysis when expression match with desired pattern. The few constraints can also be placed on the condition of system artifacts. Different pattern specification can be written even to extract artifacts from text file having associations in different data attributes. The nested specifications can also be used to extract the required artifacts.

The regular expression patterns designed by the other programmers become difficult by the novice users to understand. So we can use comments in the regular expression syntax to explain the specification of patterns as shown in the following pattern specification.

**Pattern** (?#comments)\{(.*)\}.

TABLE II
EXTRACTED FROM PASCAL CODE

```
.+commandhandling.pas  20    procedure
HandleNewCommand;
**      22    procedure InstallAppCommands;
**      28    procedure andleAbout(theWindow
: WindowRef);
**      34    procedure HandleNewCommand;
**      130    procedure InstallAppCommand
```

In our case study our concentrated on Cobol legacy code because still industry is converting the legacy systems of Cobol into new software applications. The following pattern specification is used to extract the Cobol file name from Source code of (Human Resource Program) developed in Cobol.

**Pattern:** FD\s+\w+

We can also extract the complete file and record structures from the source code of COBOL by different pattern specifications. These specifications further may be used for recovering the ERD model of COBOL applications.

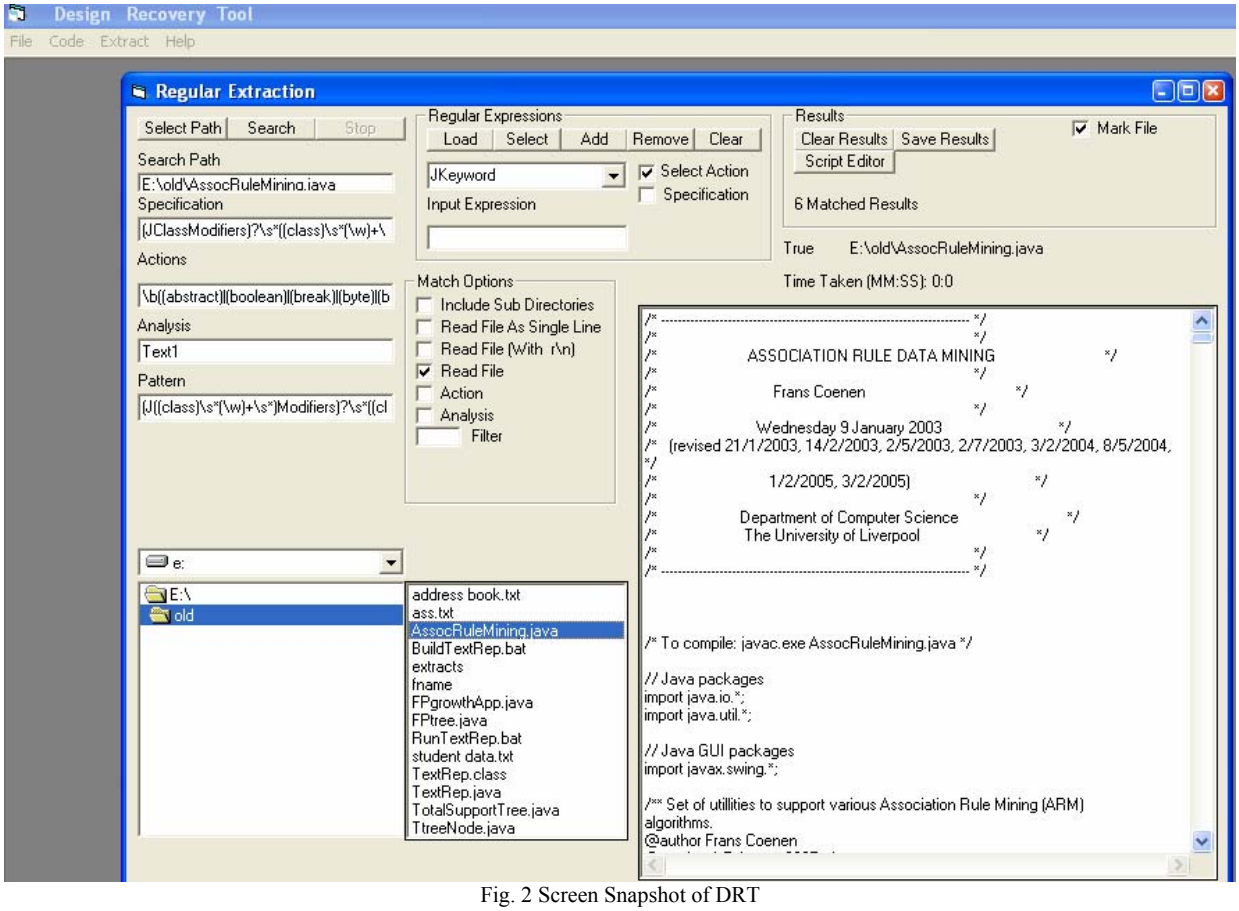

Fig. 2 Screen Snapshot of DRT

We use the regular expression specifications at different abstraction levels to extract the desired artifacts. For example the following pattern specifications are used to extract Java classes from the source code of Java applications.

**Pattern1**: ((class)\s*(\w)+\s*\{)

Pattern1 will extract only classes without extends and implements functions of Java classes. We use the above pattern specification to further extract the derived classes with extends and implements arguments in pattern 2.

**Pattern2**:

**(JClasModifiers)?\s*((Class)((extends)\s*(\w)+)?\s*((implements)\s*(\w)+)?(\s*(,)\s*(\w+))*\s*\{)**

In pattern2 the definition of JClasModifiers is abstracted. Similarly we can use lower to higher level of abstractions to extract our desired artifacts. The specifications are also designed to represent the relationships between the extracted artifacts which are further used for recovering architecture of different software systems.

Similarly we can write following pattern specification to extract all procedures, functions and property procedure from source code of Visual basic (Monica database application ) as shown in Fig 3.

**Pattern**: (VBproc|VBfun|VBprop)

The pattern specifications of VBproc, Vbfun, Vbprop are as given below in Pattern 1a, Patten 1b and Pattern 1c.

**Pattern1a** : ((Private|Public)\s*)?\s*(Static)?\bSub\b\s*(\w+)

**Pattern1b:**
((Private|Public)\s*)?\s*(Static)?\bFunction\b\s*(\w+)

**Pattern1c**:
((Private|Public)\s*)?\s*(Static)?\s*Property\s*(Get|Let|Set)\s*(\w+).

The legacy systems may have source code of million lines. The artifacts extraction speed is concerned while extracting artifacts from large systems. The Table III shows the time taken by our tool for extracting artifacts from Tacacs source code[27].

TABLE III
SCANNING TIME

| Tacacs Source Code | Time Taken | No of artifacts Extracted |
|---|---|---|
| Scanning complete code | :57 | 19987 |
| Include Files | 0:0 | 153 |
| Function calls | 0:0 | 242 |
| Comments | 0:1 | 1708 |

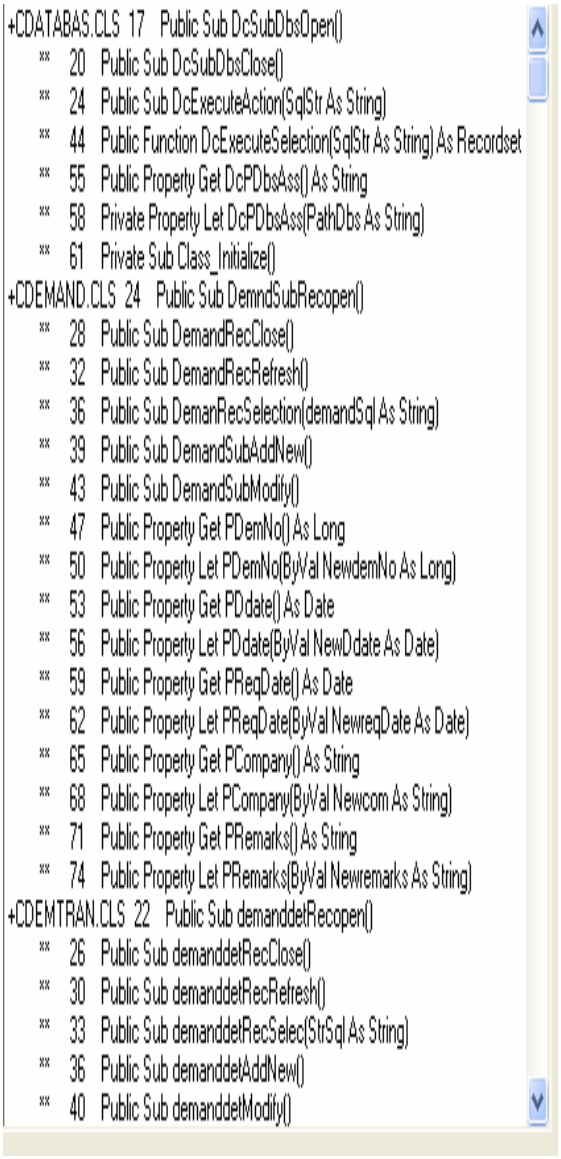


Fig. 3 Extracted procedures

## IV. CONCLUSION

Reverse Engineering will always be necessary and play important role for recovery of knowledge from legacy systems. The proposed Architecture Recovery Framework is an attempt to combine application domain knowledge, architecture recovery approaches and tools in order to recover the software architecture of legacy systems. The Recovery framework is used on code and document of five different programming language and has successfully recovered different desired artifacts with the help of recovery process and tools. The regular extraction technique is used to extract the artifacts at various abstraction levels.

## V. Future Work

Future work consist of building the tools for process automation, application of process and framework to large and complex software systems and refinement of process and framework based on experiences and integration with different development processes. The proposed framework will be tested with different large and complex software systems using different recovery approaches and tools.

## References

[1] Bass, Clements, Kazman, *Software Architecture in Practice,* 2nd ed. Addison-Wesley, 2003.
[2] Gall H., Jazayeri M, Klosch R, Lugmayr W., Trausmuth G., "Architecture Recovery in ARES", in *Proc. of the 2nd International Software Architecture Workshop* (ISAW-2), San Francisco, 1996.
[3] Nadim Asif, "Architecture Recovery", In the Proc. of International Conference of Information and Knowledge Engineering (IKE'02), las Vegas, 2002.
[4] David Garlin, "Research directions in software architecture" ACM Computing surveys, Vol. 27. No. 2, 1995. pp. 257 – 261.
[5] O'Brien, L., "*Dali*: A Software Architecture Reconstruction Workbench", Software Engineering Institute, Carnegie Mellon University, May 2001.
[6] P.J. Finnigan, R. Holt, I. Kalas, S. Kerr, K. Kontogiannis, et al. The software bookshelf. IBM Systems Journal, 36(4):564–593, November 1997.
[7] Imagix4D,UR=http://www.imagix.comImagix Corp. [Accessed 10th march, 2007].
[8] Bauhaus group, "Tour de Bauhaus", http://www.*Bauhaus*-stuttgart.de/demo/index.html, version 4.7.2, December 2003.
[9] Wolfgang Eixelsberger ReneKlosch, "A Framework for Software Architecture Recovery", http://www.sei.cmu.edu/productlines/splc/las_navas/14eixelsberger.pdf. [Accessed on March, 2007].
[10] Nadim Asif, Mark Dixon, Janet Finlay, George Coxhead,"Recovering the Design artifacts", In *Proc. of International Conference of Information and Knowledge Engineering* (IKE'02), Las Vegas, 2002.
[11] Burnstein and K. Roberson, "Automated Chunking to support program comprehension", in the *Proceeding of IWPC*' 97, Michigan, 1997, pp. 40-49 .
[12] K. Erdos, H.M Sneed, "Partial Comprehension of Complex Programs" I in *Proc. of 6th International Workshop on Program Comprehension*, Ischia, Italy, 1998.
[13] Jahnke J.H, Walenstein A., "Reverse Engineering tools as media for Imperfect Knowledge" in *Proc. of Working conference on Reverse Engineering* (WCRE00),Queensland, Australia, 2000.
[14] R.Keller, R.Schauer, S.Robitaille, P.Page, "Pattern Based Reverse-Engineering of Design Components". In *Proc. 21st International Conference on Software Engineering*, 1999.
[15] Jianjun Zhao, "Slicing Concurrent Java Programs", in *Proc. of IEEE – 7th International Workshop on Program Comprehension*, 1999, pp. 126-133.
[16] Gannod G.C, Cheng B.H.C, "As Formal Approach for Reverse Engineering," in *Proc. of IEEE Proceedings of Working Conference on Reverse Engineering*, 1999.
[17] A. Von MayrHauser and A.M. Vans, "Program Comprehension during Software Maintenance and Evolution", IEEE Computer, Vol. 28. pp 44-55, August 1995.
[18] J.M. DeBaud, B. Moopen, S. Rugaber, "Domain Analysis and Reverse Engineering," http://www.cc.gatech.edu / reverse/papers.html [accessed on March, 2007]
[19] Kamran Sartipi "Software architecture recovery based on pattern matching", PhD Dissertation, University of Waterloo Canada, 2002.
[20] Hongji Yang, *Software evolution with UML and XML* Idea Group Publishing, 2005, page 58, 2004.
[21] Onaiza Maqbool, "Architecture recovery of legacy systems", PhD thesis, univerisity of lahore pakistan, September 2006.
[22] Rainer Koschke, Gerardo Canfora, Jörg Czeranski, "Revisiting the ΔIC approach to component recovery", Science of Computer Programming, Volume 60, Issue 2 (ISSN:0167-6423 ), Pages: 171 - 188 , April 2006.
[23] Rainer Koschke, Gerardo Canfora, Jörg Czeranski, "Revisiting the ΔIC approach to component recovery", Science of Computer Programming, Volume 60, Issue 2 (ISSN:0167-6423 ), Pages: 171 - 188 , (April 2006).
[24] Rigi,URL=http://www.rigi.csc.univ.ca/rigi/rigiindex.htm .l [Accessed 5th march, 2007].
[25] Nadim Asif , Reverse Engineering Methdology to Recover the Design Artifacts: A Case study, In *Proceedings of International Conference of Software Engineering Research and Practices* (SERP'03), 23rd-26th June 2003, Las Vegas, USA, CSREA Press, pp.932-938.
[26] Nadim Asif, Recover the Use Case Models, In proceedings of International Conference of Software Engineering research and Practice (SERP05), 27th-30th June, 2005 Las Vegas, USA, CSREA Press.
[27] http://www.gazi.edu.tr/tacacs/ [Accessed 10th June 2007].

**Ghulam Rasool,** is a faculty member at a prestigious public sector (state owned) institute, COMSATS Institute of Information Technology, Lahore. For a number of years Mr Rasool is delivering his valuable knowledge in the field of Software Engineering. He did his Masters in Computer Science from Bahauddin Zakariya University, (A public sector University) in Multan, Pakistan. Currently, he is conducting extensive research for his Masters of Science Thesis in the area of Software Engineering, under the supervision of Prof. Dr. Nadeem Asif. Rasool has been heavily involved in a number of projects and in teaching for the subjects of MIS, Software Engineering and Data Structures For the last eight years.

**Dr. Nadeem Asif**, is a Professor in Software Engineering at Department of Computer Science, The University of Lahore, Pakistan. He is also heading the department for some years. Prof. Asif completed his Ph. D from University of Leeds, UK. His work has been published in a number of re-known conferences and journals. Beside research, he has worked as Co-Editor, Editor, Associate-Editor and Program Committee Member of a number of International Conferences. Nevertheless he is the Editor-in-Chief of a highly reputed journal on software engineering. At University of Lahore he has organized international conference/s.

Prior to joining he has worked, as head of department at Department of Computer Science, National College of Business Administration and Education (NCBA&E), Lahore. Currently, at University of Lahore, Dr Asif is supervising a number of Master's of Science Students for their Master's thesis.