EXTRACTION OF CONNECTOR CLASSES FROM OBJECT ORIENTED SYSTEM WHILE RECOVERING SOFTWARE ARCTITECTURE

A thesis submitted to

Tilak Maharashtra University, Pune

For the Degree of Vidyavachaspati (Ph.D.)

in the

Computer Management

Under the Faculty of Management

By

Mrs. Shivani Budhkar

Under the Guidance of

Dr. Arpita Gopal
Director-MCA
Sinhagad Institute of Business Management and Research,
Kondhwa, Pune - 411048
July 2013

I hereby declare that the thesis entitled **"Extraction of connector classes from object oriented system while recovering Software Architecture"** completed and written by me has not been previously formed the basis for the award of any Degree or other similar title upon me of this or any other university or examining body.

Place: Pune (Shivani Budhkar)

Date: 30th July 2013 Research Student

Certificate

This is to certify that the thesis entitled **"Extraction of connector classes from object oriented system while recovering Software Architecture "** which is being submitted for the degree of Doctor of Vidyavachaspati (Ph.D.) in Computer Management to Tilak Maharashtra University is an original piece of research work completed by Mrs. Shivani Budhkar under my supervision and guidance.

To the best of my knowledge and belief the work incorporated in this thesis has not been formed the basis for the award of any Degree or similar title of this or any other university or examining body upon her.

Place: Pune

Date: 30th July 2013

Dr. Arpita Gopal

Director-MCA

Sinhgad Institute of Business Administration

and Research, Kondhwa-Bk, Pune – 411048

Acknowledgement

It would not have been possible to write this doctoral thesis without the help and support of the kind people around me, to only some of whom it is possible to give particular mention here.

I would like to express my deepest gratitude to Dr. Arpita Gopal, my research supervisor, for her patient guidance, valuable support, enthusiastic encouragement, useful critiques of this research work and assistance in keeping my progress on schedule.

My heartfelt thanks go to my fellow research scholar and friend, Chandrani Singh, whose valuable support and help kept me on track.

I am particularly grateful for the assistance given by my friend Aniket Gujarathi. I would like to thank him, who as a good friend was always willing to help and give his best suggestions.

I would also like to thank my parents, my younger sister, and brother in law. My parents are always big source of inspiration for me. They were always supporting me and encouraging me with their best wishes for which my mere expression of thanks likewise does not suffice.

I deeply grateful to my husband, Mr.Ashutosh Budhkar for constant support and sharing the household responsibilities with me to let me have time to focus on my work. He was always there cheering me up and stood by me through the good times and bad. To my sweet little daughter, Devashree, who was always excited to see my research work and

was happy to see my research publications. She never disturbed me when I was doing my research work at home and did her studies at her own.

To my in laws who are very excited to see my completed research study. Their constant encouragement and blessing were always with me during my research work. I express my deepest gratitude.

I am indebted to Dr. Prof. K.R.Joshi, Principal Modern College of Engineering, Pune, who always has a positive attitude towards academic, and research endeavours and provided me with ample opportunities to work and explore her academic leadership, and quest for excellence have always been a source of inspiration. I would also like to thank Dr. Prof. Desai A.D., vice- principal, Modern College of Engineering, Pune, Dr. Ekbote G.R. Chairman, P.E.Society, Pune for their valuable support.

Finally I would like to thank all my friends who encouraged me during my research work. Thank you all.

Shivani Budhkar
Research student

Contents

a. Component evaluation and component interface identification from object oriented System by Shivani Budhkar, Dr. Arpita Gopal, in International Journal of Advanced Research in Computer Science, ISSN No. 0976-5697, Volume 3, No. 4, pp. 84-90,July- August 2012

b. Component identification from existing object oriented system using Hierarchical clustering by Shivani Budhkar, Dr. Arpita Gopal, in IOSR Journal of Engineering, ISSN: 2250-3021, Vol. 2(5) pp: 1064-1068, May. 2012

c. Component based software architecture recovery from object oriented system using existing dependencies among classes by Shivani Budhkar, Dr. Arpita Gopal,in International Journal of Computational Intelligence Techniques ISSN: 0976-0466 & E-ISSN: 0976-0474, Volume 3, Issue 1, 2012, pp.-56-59, April 2012

d. Reverse Engineering Java Code to Class Diagram: An Experience Report, by Shivani Budhkar, Dr. Arpita Gopal, in International Journal of Computer Applications (0975 – 8887) Volume 29– No.6, pp. 36-43, September 2011

e. Component interactions from software architecture recovery by Shivani Budhkar, Dr. Arpita Gopal, in International Journal of Computer Science and Communication Vol. 2, No. 1, pp. 149-15, January-June 2011

f. Extraction of Connector Classes from Object –Oriented System while recovering Software Architecture by Shivani Budhkar, Dr. Arpita Gopal, in IEEE International Advance Computing Conference (IACC 2009) Patiala, India, pp.1826-1828, 6–7March 2009

List of Figures

List of Tables

Chapter 1

Introduction

Computing environments are evolving from mainframe systems to distributed system. Many legacy systems today are object oriented. Today in addition to object oriented techniques in software development, components are used. Development of distributed systems is more and more based on the use of component technology. Components are regarded as being more course-grained compared to traditional reusable artifacts such as objects and provide high level representation of the domain. Components can be used more effectively and are better suited for reuse than using objects. Maintainability and reliability of software is improved by reusing existing elements / components. Hence, we should derive reusable components and connectors from classes in object oriented systems and change object oriented systems into component based systems. These component based systems are suitable for distributed systems and multiple systems can make use of these components and connectors.

Software reuse is one of the most researched subjects in software engineering. Software reuse is the process of implementing and / or updating software systems using existing software assets. This results in improved software quality and productivity. This in turn reduces the time to market.

According to Suk Shin et al [91] Component based development is an effective reuse technology which extensively utilizes object oriented design; therefore, it is economical approach to generate component based design form object oriented design.

Component based software development extends object oriented software development paradigm. It assembles and reuses pre- existing software components. Creating new Component based development cost is higher than conventional software development .It is better to use existing object oriented code to create components and migrate into component based system.

One of the most prominent maintenance objectives is migrating systems to distributed computing environments using components.

To maintain and understand large applications, it is crucial to know software architecture. Software Architecture plays very important role in all phases of software development. Most of the existing systems do not have reliable software architecture and some legacy systems are designed without software architecture design phase. Thus software architecture recovery is very important task. Reverse engineering will always be necessary and play important role for software architecture recovery from the existing software. So, by doing reverse engineering, we can retrieve component based software architecture from existing object oriented software. Component based software architecture is beneficial as it is useful for reusing system parts represented as components. The software architecture of the system is described as a collection of components along with the interaction among these components, where as the main system functional block are components, they strongly depend on connectors – which is abstraction capturing nature of these interactions.

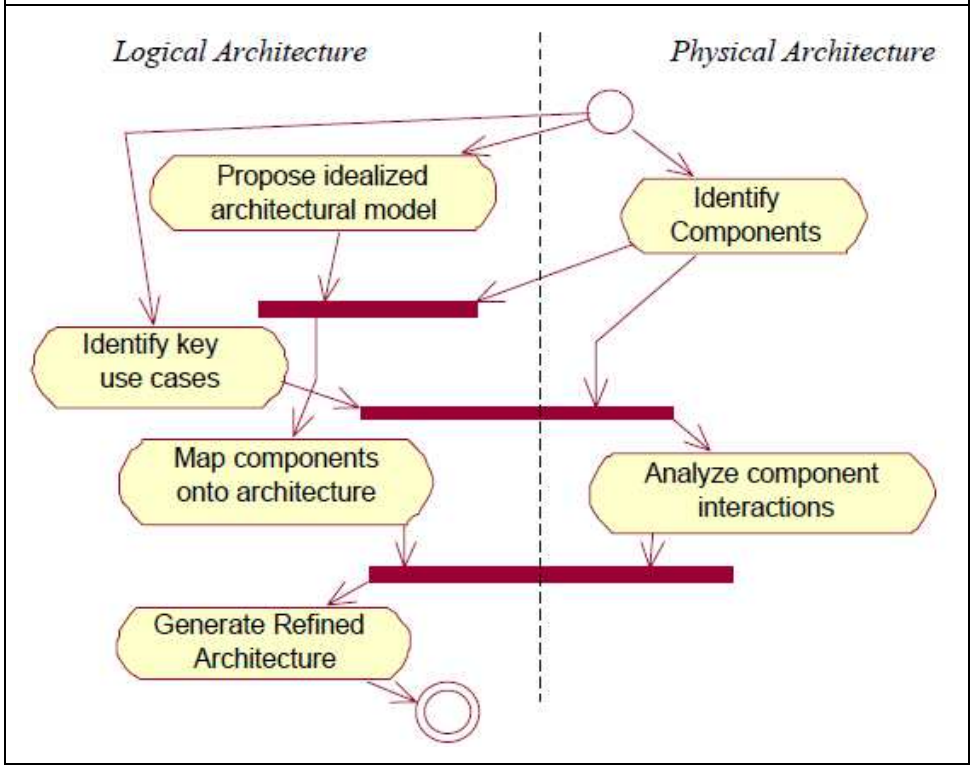
Therefore we should derive reusable components and connectors from classes in object oriented system and change object oriented system into component based system suitable for distributed environment where many systems make use of components and connectors.

1.1. Software Architecture Recovery

Software architecture definitions in general contain three elements: components, connectors and rationales, such as e.g. 'The structure of the components of a program/system, their interrelationships, and principles and guidelines governing their design and evolution over time' described by Wolfgang et al [100]. Gall et al [23] defined architecture recovery as a process of identifying and extracting higher level of abstractions from existing software systems. Architecture recovery and reengineering to handle legacy code is critical for large and complex systems. Software architecture recovery is a set of methods for the extraction of architectural information from lower level representations of a software system, such as source code. The abstraction process to generate architectural elements frequently involves clustering source code entities (such as files, classes, functions etc.) into subsystems according to a set of criteria that can be application dependent or not. It is described in Wikipedia [112] that Architecture recovery from legacy systems is motivated by the fact that these systems do not often have an architectural documentation, and when they do; this documentation is many times out of synchronization with the implemented system. Alae-Eddine et al [3] defined Component-based software architecture as a high level abstraction of a system using the architectural elements: components which describe functional computing, connectors which describe interactions and configuration which represents the topology of connections between components.

The figure 1.1 shows architecture recovery steps using FOCUS approach proposed by Nenad Medvidovic and Vladimir Jakobac[55]. This is light weight approach for recovering and evolving architectures of undocumented Object oriented applications.

Figure 1.1: Architecture Recovery Steps of FOCUS



1.2. Issues in Component Based Software Architecture Recovery

- There is a distinct lack of a complete methodology for reengineering an object oriented legacy system into system that consists of components described by Eunjoo Lee Byungjeong [18].

- Object oriented design (OOD) can be transformed into Component –based design (CBD) suggested by Suk Shin [91]. For this approach one can have object oriented design specification available, which is mostly not available for legacy systems.

- According to Mishra et al [74] there are various approaches which deal with partial recovery of component based architecture i.e. only reusable components are identified

through reverse engineering. Using weighted directed graph and hybrid clustering algorithm, object oriented software is partitioned to groups as components, described by Qifen et al [66] but no details about interfaces between classes or components,.

- Simon, Houari et al [83] and Aline et al [1] used executing execution traces which are generated by using use cases as dynamic dependencies to identify components from object oriented system. Then using global search (genetic algorithm), and local search (Simulating Annealing algorithm) components are defined. For this approach, system use cases are needed. If no documentation is available for use cases, it becomes difficult to start with.

- Formal concept analysis technique can also be used to identify methods shared by use case implementation. Each concept in the generated conceptual lattice encompasses a set of use cases and their shared methods. However, the lattice does not make clear where a source code entity, such as class, must be located in the architecture, since same entity appears in more than one concept, suggested by Thomas Tilley et al [95].

- Some of the approaches which support dynamic analysis for component based software architecture, described by Lei Ding and, Nenad Medvidovic [48], which is not automated.

- For many existing legacy systems, software architecture representation is not available. This is required in every phase of software, mostly in software maintenance phase and migrating to new technology. Cost wise it is beneficial to reuse existing source code rather than developing entire new system.

- Most of the transformations from object oriented system to component based systems require lots of human experts like designers of old system, maintainers, user etc. Fully automated approaches are very less in this kind of transformation.

1.3. Approaches towards Software Architecture Recovery

Several excellent approaches and techniques have been proposed in literature to support software architecture recovery. Sylvain Chardigny et al [93] distinguish these works according to process input used, approach and techniques used to extract architecture.

1.3.1 Inputs For Software Architecture Recovery:

The various works proposed in the literature have various inputs for software recovery process. Inputs used for software architecture recovery can be of two types: Non architectural input and architectural input.

- **Non architectural input :** Pollet et al [64] suggested non architectural inputs are source code e.g. RMTool, symbolic textual information available in comments or in the method names e.g. Anquetil and Lethbridge recover architecture from the source file names, dynamic information like run time events such as method calls, CPU utilization, network bandwidth, physical organization of application in terms of files and folders often tells architectural information ManSART and Softwarenaut work from the structural organization of physical elements such as files, folders, or packages. Some approaches map packages or classes to components and use the hierarchical nature of the physical organization as architectural input. It is then important to consider the influence of the human organization on the extracted architectures or views. Bowman et al [39] used the developer organization to form an ownership architecture that helps stakeholders reconstruct the software architecture. According to Pollet et al [64] non architectural information like historical information is rarely used in software architecture recovery. For example ArchView is a recent

approach that exploits source control system data and bug reports to analyze the evolution of recovered architectural views. To assist a reverse engineer in understanding dependency gaps in a reflexion model, Hassan and Holt, Murphy annotates entity dependencies with sticky notes. These sticky notes record dependency evolution and rationale with information extracted from version control systems. ArchEvo produces views of the evolution of modules that are extracted from source code entities. Human expertise as non architectural information is very helpful when it is available. At high abstraction levels, Software architecture recovery is iterative and requires human knowledge to guide it and to validate results. To specify a conceptual architecture, reverse engineers have to study system requirements, read available documentation, interview stakeholders, recover design rationale, investigate hypotheses and analyze the business domain. Human expertise is also required when specifying viewpoints, selecting architectural styles, or investigating orthogonal artifacts. While software architecture recovery processes involve strategy and knowledge of the domain and the application itself, only a few approaches take human expertise explicitly into account. Ivkovic and Godfrey [36] proposed to systematically update a knowledge base that would become a helpful collection of domain-specific architectural artifacts.

Most often it works from source code representation but it also considers other kinds of information. Most of them are non-architectural. For example - human expertise used in interactive way in order to guide the process. Some works use architectural input like style. For example Focus approach proposed by Lei Ding and, Nenad Medvidovic [48] uses style in order to infer a conceptual architecture that will be mapped to a concrete architecture extracted from source code. Some uses documentation as input along with source code. According to Pollet et al RMTool [64] directly query the source code using regular expressions as non architectural inputs. Finally most works are based on human expertise: Some use expertise of architect which uses tool as input.

- **Architectural inputs:** Architectural inputs can be architectural style and viewpoints.

  Style: Architectural styles such as pipes and filters, layered system, data flow are popular because like design patterns, they represent recurrent architectural situations. They are valuable, expressive, and accepted abstractions for software architecture recovery and more generally for software understanding. Examples of architectural styles are pipes and filters, blackboard, and layers. Recognizing them is however a challenge because they span several architectural elements and can be implemented in various ways. The question that turns up is whether software architecture recovery helps reverse engineers specify and extract architectural styles suggested by Pollet et al [64]. For Examples: In Focus, Ding et al [48] use architectural styles to infer a conceptual architecture that will be mapped to a concrete architecture extracted from the source code. Medvidovic et al [54] introduce an approach to stop architectural erosion Their approach considers architectural styles as key design idioms since they capture a large number of design decisions, their rationale, effective compositions of architectural elements, and system qualities that will likely result from using the style.
  Viewpoints: The system architecture acts as a mental model shared among stakeholders. Since the stakeholders' interests are diverse, viewpoints are important aspects that software architecture recovery may consider. Viewpoint catalogues were built to address this issue: the 4 + 1 viewpoints of Kruchten; the four viewpoints of Hofmeister et al [7], the build-time viewpoint introduced by Tu and Godfrey or the implicit viewpoints inherent to the UML standard. Pollet et al [64] described that most software architecture recovery approaches reconstruct architectural views according only to a single or a few preselected viewpoints. For Examples: The Symphony approach of Van Deursen et al [98] aims at reconstructing software architecture using appropriate viewpoints. Viewpoints are selected from a catalogue or defined if they don't exist, and they evolve throughout the process. They constrain SAR to provide architectural views that match the stakeholders' expectations, and

ideally are immediately usable. The authors show how to define viewpoints step by step, and apply their approach on four case studies with different stakeholder goals. They provide architectural views to reverse engineers following the viewpoints, these reverse engineers typically use during design phases. Pollet et al [64] described that Riva proposed a view-based SAR approach called Nimeta based on Symphony: Nimeta is a full SAR approach that uses the Symphony methodology to define viewpoints.

- **Mixed inputs:** Most approaches work from a limited source of information, even if multiple inputs are necessary to generate rich and different architectural views. Kazman et al [68] advocate the fusion of multiple sources of inputs to produce richer architectural views: for example, they produce inter-process communication and file access views. Lange and Nakamura [17] mix dynamic and static views to support design pattern extraction. Pollet et al [64] described ArchVis uses source code, dynamic information such as network log or messages sends and file structures. Multiple inputs must be organized and Ivkovic and Godfrey [37] proposed a systematic way to organize application domain knowledge into a unified structure.

1.3.2 Software Architecture Recovery Based on Approaches used:

Software Architecture Recovery processes classified based on their flow to identify architecture: bottom-up, top down or hybrid.

- Bottom-up approach

In this approach we start with low level knowledge like source code and gradually discover the complete architecture. Several tools support this bottom-up process.

The Dali tool by Rick et al. [69] [70] supports a typical example of a bottom-up process: (1) Heterogeneous low-level knowledge is extracted from the software implementation,

treated and stored in a relational database. (2) Using the Rigi visualization tool by Hausi et al [30], a reverse engineer visualizes and manually abstracts this information. (3) A reverse engineer specifies patterns by selecting source model entities with SQL queries and abstracting them with Perl expressions. Based on Dali, Guo et al [26] proposed ARM which focuses on design patterns conformance.

Other examples of bottom-up approaches include ArchView, Revealer and ARES, ARMIN Gupro described by Pollet and Ducasse [64]. Also ROMANTIC approach proposed by Chardigny, et al [93] is bottom up approach.

- Top-down approach

In this approach we first build conceptual architecture of system in terms of some pattern. The software system is then searched to find instances of that pattern. Conceptual architecture is formed with the help of requirements or architectural styles.

Top-down processes start with high-level knowledge such as requirements or architectural styles and aim to discover architecture by formulating conceptual hypotheses and by matching them to the source code. The term architecture discovery often describes this process. For example Reflexion Model of G. Murphy [20] is a typical example of Top- Down process. In this model reverse engineers first defines his high-level hypothesized conceptual view of the application then he specifies how this view maps to the source code concrete view. The reverse engineer iteratively computes and interprets reflexion models until satisfied.

- Hybrid approach

This approach is a combination of the previous two – Bottom-up and Top-down. On one hand, low level knowledge is abstracted up using various techniques. On the other hand high level knowledge is refined.

For example, Igor et al [36] proposed a hybrid architecture recovery methodology called Dynamo-I, which recovers conceptual architecture based on documentation available. It also identifies key use cases by analyzing user level behavior of the application. The approach also uses source code of the application for recovering of software architecture.

Tzerpos et al [97] presented a hybrid process in which they combined extracted code facts and information derived from interviewing developers to determine the architectural structure of a legacy system. This approach is combination of the classic top down and bottom up approaches. The approach is based on experience with large industrial application.

FOCUS proposed by Ding and Medvidovic [48] also uses hybrid process. Other hybrid processes are Nimeta, ManSART, ART, X-Ray, ARM and DiscoTect described by Pollet and Ducasse [64].

As with any classification, the borders are fuzzy for these categories.

1.3.3 Software Architecture Recovery Based on Techniques used:

The research community has explored various techniques to reconstruct architecture that can be mainly classified according to their automation level.

- Quasi-manual

The reverse engineer manually identifies architectural elements using a tool to assist him to understand his findings. There are two categories of this technique namely: Construction based techniques and Exploration based techniques. Construction based techniques reconstruct the software architecture by manually abstracting low level knowledge e.g. Rigi, CodeCrawler described by Pollet et al [64].Exploration based techniques give reverse engineers an architectural view of the system by guiding them

through the highest-level artifacts of the implementation, like in Softwarenaut by Mircea Lungu et al [51]. The architectural view is then closely related to the developer's view.

Another example of quasi-manual technique is, Focus by Ding and Medvidovic [48] regroups classes and maps the extracted entities to an idealized architecture obtained from an architectural style according to the human expertise. It is one of the bottom-up approaches, where it is assumed that little or no documentation is available for system modification. In addition to this, the basic architecture of the original system and desired properties of the application are assumed to be known.

- Semi-automatic

It automates repetitive aspects of the extraction process but reverse engineer steers iterative refinement or abstraction, leading to the identification of architectural elements. That is the reverse engineer manually instructs the tool how to automatically discover refinements or recover abstractions.

For example, in Dali reverse engineer specifies reusable abstraction rules and execute them automatically using SQL described by Sylvain et al [93].

Some approaches build analyses as plain object-oriented programs. Stéphane Ducasse et al described [89] For example; the groupings made in the Moose environment are performed as object-oriented programs that manipulate models representing the various inputs.

- Quasi-automatic

Pure automatic techniques for reconstructing the software architecture tend towards automatic process but still reverse engineer must steer them. Concept, dominance and cluster analysis are the techniques which are often combined for software architecture recovery in quasi-automatic techniques.

Concepts: Formal concept analysis is a branch of lattice theory used to identify design patterns, features or modules. Tilley et al [95] present a survey of work using formal concept analysis.

Clustering Algorithms: Clustering algorithms identify groups of objects whose members are similar in some way. They have been used to produce software views of applications. To identify subsystems, Anquetil and Lethbridge [56] cluster files using naming conventions. Some approaches automatically partition software products into cohesive clusters that are loosely interconnected suggested by Spiros et al [87] and Theo Wiggerts et al [94]. Maher Salah [49] described that Clustering algorithms are also used to extract features from object interactions.

Dominance: In directed graph, a node D dominates a node N if all paths from a given root to N go through D. In software maintenance, dominance analysis identifies the related parts in an application .Lundberg and Löwe [44] outline a unified approach centered around dominance analysis. On the one hand, they demonstrate how dominance analysis identifies passive components. On the other hand, they state that dominance analysis is not sufficient to recover the complete architecture: it requires other techniques such as concept analysis to take component interactions into account.

Recent example of quasi-automatic approach is ROMANTIC approach developed by Chardigny et al [93]. It is also bottom-up approach which uses other semantic information about the system like architecture elements, architectural quality to extract architecture in addition to source code and decreases the need of human expertise.

Even though software architecture recovery works are classified according to process input used, approach and techniques used to extract architecture, but the process of software architecture recovery depends on what are the stakeholders' goals; how does the general reconstruction proceed; what are the available sources of information, based on

this entire software architecture approach is decided, and finally what kind of knowledge does the process provide.

1.4 Research Hypothesis

The proposed work is aimed at providing assistance to software maintenance for transforming existing object oriented system to component based system. Thus, reusing existing code and migrating to new environment saves cost, efforts of redesign and redeveloping the system which suits to new evolving environment. This is what the software industry always prefers.

This research work endeavors to achieve the following

- To develop approach and tool for migration from object oriented system to component based system.

- The tool will assist to extract components and interface details from object oriented system to form component based system.

- Maximum automation and less human intervention will reduce human efforts and cost of software development by reusing existing object oriented system instead of starting development from scrap.

- Extracted components will also be evaluated by tool for quality monitoring using metrics like size of component, coupling of component and cohesion within component.

1.5 Research Methodology adopted

The proposed research work is divided into 3 steps

Before using proposed approach and tool, use any UML reverse engineering tool to generate class diagram, which will help to compare the results from step-I. To extract

classes from existing object oriented system, any good quality reverse engineering tool like IBM's Rational Rose or Enterprise architecture can be used .By experiments find which tool extracts maximum and accurate information about classes from object oriented system.

Step – I Component Based Architecture Recovery from Object Oriented System from Existing Dependencies among Classes -

The proposed approach is based on the identification of source code entities and the relationship between them. The list of possible relationships between object oriented systems includes inheritance, composition, invocation relationship etc. Also these dependencies are used to generate input needed for next step i.e. identify components. Thus this step consists of

- Existing java source code from folder is input to the tool
- Identify and display dependencies among classes like inheritance coupling, composition coupling, method coupling and integrated coupling of them in tabular format.

Step –II Component Identification from Existing Object Oriented System using Hierarchical Clustering -

- Using identified dependencies and clustering algorithm, cluster levels will be formed and components will be defined.
- We will propose agglomerative hierarchical Clustering algorithm for this step. Input for the algorithm is taken from the previous step i.e. dependencies among classes

Step-III Component Evaluation and Component Interface Identification from Object Oriented System -

Identified group of classes working together will form components. Using the components created in previous step interface details will be identified and components will be evaluated for quality using component quality metrics. The interface details can be bundled into packages, which will act as connector between the components. Thus this step consists of

- Identify interface details for the components created in the previous step.
- Evaluate the components for quality monitoring using metrics like size of component, coupling of component and cohesion within component.

Once components are evaluated, interface details are extracted and the component based software representation is ready.

The above mentioned methodology will be simulated on a java application. The study is specific to Java object oriented source code but gives general idea about proposed tool and entire approach.

This work is not proposes for deploying components and connectors. It is assumed that software maintenance person knows how to deploy using the component based framework or model, the organization uses. For example, if a software company uses OSGi model, then extracted interface details can be bundled into package which can be imported and exported as per requirement. So classes and interfaces play a role of required and provided interfaces.

The software maintenance person can use the extracted details from proposed approach and using his or her knowledge can rewrite components and connectors by giving names to them. For example, the tool extracts components with the names e.g. Component0, Component1, Component2 etc. He should rename these at the time of implementation like in ATM system component-bank, component- transaction etc. Thus user should be able to create packages with the component name.

1.6 Theoretical and Practical Significance of Proposed Work

Theoretically this research will contribute to the existing component based software recovery approaches from object oriented system implemented and followed in software industry.

Outcome of this research will be of practical importance to software developer and software maintenance person and to the Management of software industry for migrating the software into new computing environment by reusing existing object oriented system and reducing cost of software development with less human efforts.

1.7 Organization of Thesis

Chapter 2 presents the review of literature and the background material. First it addresses details about software architecture recovery. It gives the details of component based architecture extraction approaches and lists out their shortcomings. We have proposed the quasi- automatic approach along with its relative advantages and disadvantages.

Chapter 3 will describe which clustering algorithm type we will choose for proposed approach and why. In proposed approach, it helped us to create components from object oriented classes. The chapter presents study of existing reverse engineering tools and existing component based framework OSGi.

Chapter 4 discusses proposed entire process approach and tool. It discusses about class extraction using reverse engineering tool, identifying dependencies among the classes, clustering algorithm defined, creating inputs for the algorithm, components created, component evaluation and interface details extraction, process for creating connectors.

We have proposed agglomerative hierarchical clustering algorithm and inputs required for algorithm are generated. The process of input generation is defined in this chapter.

Quality metrics for components are proposed for evaluation of components, criteria is mentioned here so that it can be easily find that components created are of good quality. It also briefly discusses the clustering techniques, component based system, advantages of using it.

Chapter 5 gives actual implementation of tool proposed in chapter4. The chapter presents various algorithms to implement proposed tool.

Chapter 6 provides results and analysis of various experiments conducted for proposed extraction process is performed. It discusses the case study of “arithmetic24” game, developed in java, this gives guideline for user for creating components and connectors of any java object oriented system. This chapter provides a comparative study with various java application systems and comparison of the proposed approach with other existing approaches.

Chapter 7 presents summary and conclusion. It also talks about suggestions and scope for future work.

Appendix – I lists relevant definitions for understanding of fundamental about software architecture recovery.

Appendix – II (a) lists the experimental environment to implement the proposed approach and sample programs of proposed tool.

Appendix – II (b) contains overview of clustering, different kinds of clustering methods which are used for software Architecture recovery like partitional clustering algorithms and hierarchical clustering algorithm. It also describes similarity measures based on which similar clusters are grouped together.
Appendix – II (c) contains a copy of all the published papers during this research work.

Chapter 2

Review of Literature

Component based software architecture recovery from object oriented system has been handled since 1998. The present study gives thorough understanding of different approaches used to recover component based software architecture from object oriented system. The result of literature survey of these approaches is presented here.

2.1 Software Architecture Recovery

Gall H et al [23] defined software architecture recovery as a process of identifying and extracting higher level of abstractions from existing software systems. Software Architecture recovery and reengineering to handle legacy code is critical for large and complex systems. O'Brien [57] described, the recovery process can be assisted by different tools available in the market like Dali. Architecture representation consists of structural and non-structural information about software architecture. Structural information is the components and connectors describing the configuration of a system. Non structural information is architectural properties for example, safety patterns, communications patterns, behavioral patterns, structural patterns and creational patterns.

According to Garlan [25] Software Architecture plays an important role in at least six aspects of software development: understanding, reuse, construction, evolution, analysis and management. These aspects make software Architecture crucial for software development.

Stephane Kell [90] described, first problem is that architectures are not explicitly represented in code as classes as the packages are. The second problem is that software applications continually evolve and grow and so does its architecture. Hence, conceptual architecture does not match with concrete architecture.

Various works are proposed in literature in order to extract architecture from an object-oriented system. We present survey according to techniques used to extract architecture. The inputs of the extraction approaches are various. Most often it works from source code representations, but it also considers other kinds of information. Most of them are non-architectural.

2.2 Software Architecture Recovery Approaches based on techniques used

The techniques used to extract architecture are various and can be classified according to their automation level like quasi manual approaches, semi-automatic and quasi-automatic techniques.

2.2.1 Quasi manual techniques

Some methods are almost manual. These techniques construct the software architecture by manually abstracting low level knowledge and uses interactive, expressive visualization tools. Following is survey of the quasi manual approaches.

S.K.Mishra, Dr.D.S.Kushwaha, and Prof.A.K.Misra, *" Creating Reusable Software Component from Object-Oriented Legacy System through Reverse Engineering"*, 2009. In this paper authors proposed the approach Component Oriented Reverse Engineering (CORE) for development of reusable components through reverse engineering. By using the reverse engineering techniques; they extracted architectural information and services from legacy object oriented system and later on converted these services into components using OOAD(Object Oriented Analysis and Design) models like use case model, class diagram and sequence diagram. Use cases from the use case model having similar

functionalities are grouped together. They also used classes from class diagram and their relationship to identify system components. They used CRUD matrix i.e. Created, Read ,Updated and Deleted during some scenario, message-call information and class clustering for component creation. The approach is manual and time consuming. It requires some kind of automation.

Nenad Medvidovic and Vladimir Jakobac," *Using Software Evolution to Focus Architectural Recovery"*, 2006. In this paper authors proposed light weight approach Focus for recovering and evolving architectures of undocumented Object oriented applications. Architecture recovery took place in two categories logical and physical Architecture recovery. The architectures are recovered incrementally: only those parts of an application affected by a given change are modified and their architecturally relevant characteristics extensively studied and documented (hence the name "Focus"); the recovery of additional subsystems' architectures will occur only as new modifications that pertain to those subsystems are required. With each new modification, the task of recovering the architecture of the relevant subsystem and enacting the change becomes easier since a larger portion of the overall system's architecture is known and correctly documented. The approach takes the help of reverse engineering tool available in market such as Rational Rose and generates class diagram. Then some rules for grouping classes are defined by authors, using that classes are grouped together manually to form components. Once application architecture is recovered, evolution step of Focus is applied to modify application that satisfies new requirements.

Suk Kyung Shin and Soo Dong Kim ,*" A Method to transform Object oriented Design into Component based Design using Object-Z"* , 2005. In this paper authors proposed technique for transforming object oriented Design to Component based Design using Object –Z specifications. Object-Z is a formal language to design object oriented system. Using formal specifications of both OOD and CBD, they proposed set of rules to transform OOD into CBD. In this approach initially they specify key elements of OOD in

its own meta-model and then showed how OOD can be specified in object-Z. The meta-model they used of OOD is based on Object Modeling Technique (OMT).The meta-model based on OMT consists of static, dynamic and functional model. Authors then defined key elements of CBD and represented components in Component-Z which is based on Object-Z. Since there is no standard component reference model provided by OMG (Object Modeling Group), a meta- model of CBD was proposed from static, functional and dynamic viewpoints such as meta-model of OOD. Authors also specified provided and required interfaces by using some transformation rules. Resulting CBD from above approach can be implemented by utilizing object in EJB, .NET or CORBA.

Nenad Medvidovic , Alexander Egyed and Paul Gruenbacher, *"Stemming Architectural Erosion by Coupling Architectural Discovery and Recovery"*, 2003. Nenad Medvidovic et al presented approach to combine techniques for architectural discovery from system requirements and architectural recovery from system implementations. For software Architecture recovery, they generate class diagram from available tools like Rational Rose and then Classes can be grouped based on different criteria and/or architectural concerns as components. Remote procedure call (RPC) identified as connectors. In this approach the result of the recovery step is not a complete architecture of the system. Several pieces of information is still missing. This approach is not fully automated. For recovering classes form object oriented system, help from existing tools is needed.

Lei Ding and Nenad Medvidovic, *"Focus: A Light-Weight, Incremental Approach to Software Architecture Recovery and Evolution"*, 2001. In this paper Lei and Nenad proposed a guideline to a hybrid process which regroups classes and maps the extracted entities to a conceptual architecture obtained from an architectural style according to the human expertise. Authors proposed an approach called Focus, to be applied to recovering and evolving architectures of undocumented, moderately sized Object Oriented applications. Each iteration of approach is composed of two interrelated steps:

architecture recovery and system evolution. Lei Ding recovered architecture of object oriented application by proposing idealized software architectural model and then mapping it to actual component recovered. So, in this approach entire knowledge of application of which software architecture needs to be recovered should be there. Human expertise is needed for this approach.

Wolfgang Eixelsberger, Michaela Ogris, Harald Gall, Berndt Bellay,*" Software Architecture Recovery of a Program Family"*, 1998. Wolfgang et. al presented a framework for recovering the software architecture of a program family. In this framework, architectural properties such as safety or system control are recovered using different reverse engineering methods and tools in combination with architectural descriptions. The result of the architecture recovery process is the system's architectural properties and their architectural descriptions representing the architecture of a specific system. The framework was developed and applied to recover the architectures of embedded software systems. The architecture recovery framework described here is based upon four parts: - the case study, architectural properties, architectural descriptions and architecture recovery methods. These parts influence each other and limit and/or guide the architecture recovery process. As a case study authors used Train Control System (TCS) which is an embedded real time system successfully in use in different countries. The available information of case study was source code of TCS, system documentation domain knowledge engineer, application specific engineer. While working on case study authors identified several architectural properties that were not explicitly expressed in design and then enhanced them with other related properties not originally found in the case study. Each architectural property then described using one or more architectural description notations. Different architecture recovery methods are used to recover each of the previously defined architectural properties. The authors also addressed the recovery of the architecture also from structural point of view i.e. component and connector based and typically described using Architecture Description Language(ADL).

Wolfgang Eixelsberger, Lasse Warholm, Rene Klösch , Harald Gall and Berndt Bellay,*" A Framework for Software Architecture Recovery"* ,1997. In this paper authors proposed software architecture recovery framework. The input of the recovery process is the source code, the design documentation, and domain knowledge. Information from the source code can be extracted with the help of reverse engineering tools and by manual recovery. Reverse engineering tools perform static analysis on the code and extract information like call graphs, cross reference tables, and data flow diagrams. Human interaction is not possible while the tools are analyzing the source code. Manual recovery is performed on the source code by human experts, especially domain experts, can analyze the source code using their knowledge which other cannot be done by the reverse engineering tools. Thus, the framework combines application domain knowledge and the capabilities of reverse engineering tools in order to strive for the requirements of an architecture recovery tool.

2.2.2 Semi-automatic techniques

Semi-automatic methods automate repetitive aspects of the recovery process but the reverse engineer steers the iterative refinement or abstraction for identification of architectural elements. Following is survey of the semi-automatic approaches.

Aline.P.V. Vasconcelos, and C.M.L. Werner, *"Software Architecture Recovery based on Dynamic Analysis",* 2004. In this paper authors proposed an approach to software architecture recovery from object-oriented legacy systems mainly based on the dynamic analysis of systems. The process described here is iterative and incremental. The architecture is recovered in cycles, starting by the use-case modeling activity. In each cycle a more complete description of the system architecture is obtained. The process is semi-automatic and guided by a developer who must have some knowledge about the application. If developer does not have knowledge, then it has to be obtained from system experts, available system documentation and application execution. The process starts by the static reverse engineering and use-case modeling activities. The static reverse

engineering aims at the recovery of a static model of the system, which is represented through UML Class Diagrams. This activity is executed only once. The static reverse engineering was performed with Ares tool which is capable of extracting a UML static model from Java source code. Use case modeling can start in parallel. For use case modeling use cases are selected according to the change and evolution requirements of the application. Then dynamic reverse engineering starts by behavioral models such as sequence diagram. The system is executed for the specified use case scenarios and these executions are monitored, allowing the collection of execution traces. Execution traces encompass the set of events and messages generated during system execution with their sender and receiver instances and their types. To support this dynamic reverse engineering authors had developed a trace collector tool, named tracer to monitor java program executions. The approach requires domain expert knowledge.

George Yanbing Guo, Atlee, and Kazman. *"A software architecture reconstruction method"*, 1999. This paper presents semi-automatic method ARM (Architecture Reconstruction method) is an approach to architectural reconstruction distinguishing between the conceptual architecture and the actual architecture derived from source code. ARM applies design patterns and pattern recognition to compare the two architectures. ARM assumes the availability of system designers to formulate the conceptual architecture. The approach is divided into two phases:1) identification and extraction of source code artifacts, including the architectural elements and 2) analysis of extracted source artifacts to derive a view of the implemented architecture.ARM is an iterative and interpretive process; a human is integral part of the loop to evaluate the results and determine what patterns to apply in subsequent iterations.

2.2.3 Quasi-automatic techniques

Pure automatic techniques for reconstructing the software architecture tend towards automatic process but still reverse engineer must steer them. Concept, dominance and

cluster analysis are the techniques which are often combined for software architecture recovery in quasi-automatic techniques.

Following is survey of quasi-automatic techniques.

- **Software Architecture Recovery using Concepts**

Gabriela and Tom [22] described, Concept Analysis (CA) is a branch of lattice theory that allows us to identify meaningful groupings of elements (referred to as objects in CA literature) that have common properties (referred to as attributes in CA literature) 1. These groupings are called concepts and capture similarities among a set of elements based on their common properties. Mathematically, concepts are maximal collections of elements sharing common properties. They form a complete partial order, called a concept lattice, which represents the relationships between all the concepts.

Pollet et al [64] described formal concept analysis is a branch of lattice theory used to identify design patterns, features or modules. Ganter [24] described formal concept analysis is a general mathematical method for identifying commonalities within systems. It provides a way to discover sensible groupings of objects that have common attributes in a certain context ("objects" of concept analysis shall not be confused with "objects" of object-oriented programming). Informally, a concept is a collection of all the objects that share a set of attributes in a given context. The set of common attributes of the concept is called the concept's intent, and the set of objects belonging to the concept is called the concept's extent. Formally, a context is a triple C = (O, A, I), where O and A are finite sets of objects and attributes, respectively, and I is a binary relation (an instance relation) between O and A expressing the attributes each object has. The concept lattice constructed from a context describes the input on various levels of abstraction. In reengineering many approaches used formal concept analysis to identify modules and components in legacy systems. Thus concept analysis does not group items, but rather builds up so-called concepts which are maximal sets of items sharing certain features. It

does not try to find a single optimal grouping based on numeric distances. Instead it constructs all possible concepts, via a concise lattice representation.

Alae-Eddine El Hamdouni, A. Djamel Seriai, and Marianne Huchard, *"Component-based Architecture Recovery from Object Oriented Systems via relational Concept Analysis"*, 2010.In this paper authors presented approach of extracting component based architecture recovery from object oriented system using relational concept analysis(RCA).In RCA approach architectural components are identified from concepts derived by using all existing dependency relations between classes of the object oriented system. This approach is based on ROMANTIC approach developed by S. Chardigny [93]. RCA process is based on the identification of source code entities and the relations between them by source code analysis . These relations are matched with ROMNTIC refinement model. The four step RCA process is : i) Extraction of a Dependency graph (DG) of source code classes. ii) Create RCA model using dependency graph data. iii) Generate lattice of concepts representing clusters of object classes. iv) Identify candidates components from resulting lattice.

Naouel Moha, Amine Mohamed Rouane Hacene, Petko Valtchev, andYann-Ga¨el Gu´eh´eneuc,*" Refactoring of Design Defects using Relational Concept Analysis"*, 2008. In this paper authors proposed automated approach for suggesting defect-correcting refactoring using relational concept analysis (RCA). They defined a three-step RCA-based correction process that follows a two-step defect detection process. First, they build a model of the program that is simpler to manipulate than the raw source code and therefore eases the subsequent activities of detection and correction. The model is instantiated from a meta-model to describe OO programs. Next, they apply well-known algorithms based on metrics and–or structural data on this model to single out suspicious classes having potential design defects. For each suspicious class, they automatically extract a RCF that encodes relationships among class members from the model of the program. Then, the obtained RCF is fed into a RCA engine that drives the corresponding

concept lattices. Finally, the discovered concepts are explored using some simple algorithms, which apply a set of refactoring rules that allow the identification of cohesive sets of fields and methods.

Gabriela Ar´evalo, St´ephane Ducasse and Oscar Nierstrasz,*" Lessons Learned in Applying Formal Concept Analysis to Reverse Engineering,"*2005. In this paper authors used formal concept analysis to build tool to identify recurring set of dependencies for object oriented software reengineering. The approach is divided into five steps:1) Model Import: A model of the software is constructed from the source code. Moose reengineering platform, is used for these purpose , which is reengineering vehicle for object oriented software.2) FCA Mapping: A FCA Context (Elements, Properties, Incidence Table) is built, mapping from meta model entities to FCA elements (referred as objects in FCA literature) and properties (referred as attributes in FCA literature) This step is used to map the model entities to elements and properties, and they need to produce an incidence table that records which elements fulfill which property. 3) ConAn Engine: The concepts and the lattice are generated by the ConAn tool. Once the elements and properties are defined, they run the ConAn engine.The ConAn engine is a tool implemented in VisualWorks 7 which runs the FCA algorithms to build the concepts and the lattice.4) Post-Filtering: Concepts that are not useful for the analysis are filtered out. Once the concepts and the lattice are built, each concept constitutes a potential candidate for analysis. But not all the concepts are relevant. Thus they have a post-filtering process, which is the last step performed by the tool. In this way they filter out meaningless concepts. Analysis: The concepts are used to build the high level views. In this step, the software engineer examines the candidate concepts resulting from the previous steps and uses them to explore the different implicit dependencies between the software entities and how they determine or affect the behavior of the system. Thus in this paper authors presented a general approach for applying FCA in reverse engineering of object oriented software. They also evaluate the advantages and drawbacks of using FCA as a meta tool for our reverse engineering approaches.

Gabriela Ar´evalo and Tom Mens,*" Analyzing Object Oriented Framework Reuse using Concept Analysis"*, 2002. In this paper authors used the concept analysis technique to analyze classes and their methods based on their relationships in terms of inheritance, interfaces and message sending behavior. The inheritance relationship indicates whether a class is an ancestor or descendant of another one. The interface relationship indicates which methods are exported by the classes. The message sending behavior indicates which methods are called by other methods in a class. Authors calculated the concept lattice for a well-known inheritance hierarchy: the Smalltalk Magnitude hierarchy. Then, they analyzed the results after classifying the generated concepts into concept patterns. Each concept pattern allowed us to discover a number of interesting non-documented relationships (based on self sends and super sends) among classes in a hierarchy. Especially for large inheritance hierarchies, this information is crucial for understanding the software and reengineering.

Arie Van Deursen, A., Kuipers, T,*" Identifying objects using cluster and concept analysis"*, 1999. In this paper authors proposed a method for identifying objects by semi-automatically restricting legacy data structures. Authors used both Formal Concept Analysis and clustering algorithm to build Object Oriented classes from procedural source code. Elements from source code are gathered according to the features they share. Then, the resulting concepts are candidate classes and sub-concept relationships represent relations between these classes. Authors here used agglomerative hierarchical clustering algorithm. The dendrogram is prepared based on actual clusters found by the algorithm. The clustering algorithm used average linkage to measure distance between two clusters.

Houari A. Sahraoui, Hakim Lounis, Walcelio Melo, and Hafedh Mili, *"A concept formation based approach to object identification in procedural code"*, 1999. In this paper authors described migration of procedural software systems to the object-oriented (OO) technology. Their approach is based on the automatic formation of concepts, and

uses information extracted directly from code to identify objects. The approach tends, thus, to minimize the need for domain application experts. The approach is based on the relationship between data and routines. It consists of five steps. First, they compute some metrics to determine the profile of the application at hand. This profile allowed them to choose the appropriate program abstraction that they can use to identify objects. Then, they identify objects using different algorithms. Third, they identify the methods of these objects. The fourth step consists of identifying the relationships between the objects (generalization, aggregation, or more generally, associations). Finally, the source code is transformed using the so-derived object model. For object identification step they used two algorithms, a graph decomposition algorithm, and their own algorithm, which uses concept formation with Galois lattices.

Siff, M., Reps, T.W.,*" Identifying modules via concept analysis."*, 1999. In this paper, author has presented a method for identifying modules in legacy systems based on concept analysis. The entire approach is divided into three steps: - 1) Build a context, where objects are functions defined in the input program and attributes are properties of those functions. The attributes could be any several properties relating the function data structure. 2) Construct a concept lattice from the context – Concept lattice can be built from a program in such a way that concept represent potential modules. 3) Identify concept partitions. Each partition corresponds to possible modularization of input program. In this approach a formal context is built from the system elements, and both negative and positive attributes are used in order to extend the context to be well formed. Then, an algorithm of concept partition is used to discover possible partitions in the set of the generated concepts. The chosen partition represents the set of candidate classes.

- **Software Architecture Recovery using Clustering**

Clustering algorithms identify groups of objects whose members are similar in some way. They have been used to produce software views of applications. Different kinds of clustering algorithms are used in literature for software architecture recovery.

Simon Allier, Salah Sadou, Houari Sahraoui and Regis Fleurquin, "*From Object Oriented Applications to Component Oriented Application via Component Oriented Architecture"*, 2011.In this paper authors proposed a method to automatically transform an operational object oriented application in an operational component based application. The method consists of two steps: i) identify components ii) identify provided and required interfaces. For component identification step authors used traces which are identified by executing scenarios corresponding to applications use cases . Heuristic search is used to find a near –optimal solution. The static call graphs are also generated from source code. Thus using execution traces and static call graph components are created. They manually refine the components created. This approach combines two different heuristics, a genetic algorithm and simulated annealing algorithm. For second step i.e. identifying required and provided interfaces, component's services are identified by using system's call graph. These system call graphs are produced by using Variable Type Analysis (VTA) algorithm and execution traces. The identified required services are grouped together and respectively provided services according to application domain.

Siraj Muhammad, Onaiza Maqbool, Abdul Qudus Abbas, *" Role of relationship during clustering of object oriented software system"*, 2010. In this paper relationship within object oriented system are divided into different categories evaluated them for clustering process. Authors in this approach used 26 different relationships to find the similar entities, which are commonly used in the object oriented system. These relationships can be direct or indirect. The approach uses an objective function which counts the number of relationships that exists between entities (in this case classes).Greater number of relationships between two entities indicates the higher similarity between them. Thus similarity matrix is produced and hierarchical agglomerative clustering algorithm is used to cluster object oriented software system. The results produced by clustering algorithm is compared with the architecture produced manually by human experts.

Qifeng Zhang, Dehong, Qiu, Qubo Tian, Lei Sun, *"Object Oriented Software Architecture Recovery using New Hybrid Clustering Algorithm"*, 2010. Object oriented software architecture recovery using a new hybrid clustering algorithm – A Authors defined Weighted Directed Class Graph(WDCG)to represent object oriented system and then new hybrid clustering algorithm based on hierarchical clustering and partition clustering is proposed for recovering high level architecture from object oriented system. WDCG is extracted from Java byte code to represent static structure of software. They also used coupling between classes like inheritance coupling, method coupling, composition coupling , data coupling, coupling between classes , module coupling and cohesion coupling as the weights of edges. The hybrid clustering algorithms takes input WDCG , number of clusters and produced output a partition of WDCG.

Yuxin Wang, Ping Liu, He Guo , han Li, Xin Chen *," Improved Hierarchical Clustering algorithm for Software Architecture Recovery"*, 2010. In this paper authors proposed improved hierarchical clustering algorithm called LIMBO Based Fuzzy Hierarchical clustering (LBFHC) to increase the software architecture recovery accuracy and enhance the effectivity. LIMBO (ScaLable InforMation Bottleneck) algorithm proposed by Tzerpos [97] is the foundation of proposed algorithm. The LBFHC algorithm is composed of four steps: i) Identification of entities and features- For the improvement of quality and enhance cohesion of clusters , more detailed information extracted from legacy system is defined as meaningful features and associated with each entity or cluster. Different kinds of meaningful features considered here are global variables referred to by an entity , local variables referred to by an entity , user defined types used by an entity, entities called by an entity, system calls referred to by an entity , macro referred to by an entity. ii) Calculation of similarity- Based on LIMBO, information loss measure is used to calculate similarity instead of using traditional distance measure. In this case greater the information loss is , the smaller degree of similarity is . Therefore , the pair of entities or clusters , which hold the minimum value of information loss is combined into same cluster. iii)Process of clustering – The

LBFHC algorithm is presented in this step to form the clusters. iv) Selection of measures - To evaluate the clustering results for quality various measures are defined here. This step describes two primary types of measures -1) Internal evaluation It is intrinsic evaluation of clustering. It comprises the number of clusters and the percentage of arbitrary decisions, which are used to evaluate LBFHC.2) External evaluation is done with the help of expertise and experience from specialists. Both internal and external type of evaluations are compared for assessment of result from clustering.

Simon Allier , Houari A. Sahraoui and Salah Sadou*" Identifying Components in Object-Oriented Programs using Dynamic Analysis and Clustering"*, 2009. In this paper authors proposed an approach for component candidate identification as a first step towards the extraction of component-based architectures from object oriented programs. The approach used dynamic call graphs as input, built from execution traces corresponding to use cases. This approach is divided into four steps:-1) data extraction 2) possible class groups identification, 3) candidate component selection, 4) Candidate component refinement. Data (method calls) are extracted using dynamic analysis. They are obtained by executing typical use cases of the program and by grouping the corresponding execution traces into dynamic call graph (DCG).Use cases are derived from the application documentation. Using DCG concept lattice is built. The lattice's node defines group of interrelated classes. Using some heuristic selected groups are optimized. Thus resulting set of candidate components and their connections would form the component based architecture. For capturing execution traces and generating DCG, authors used existing tool, MuTT(Multithreaded Tracer). Also for constructing lattice from DCG framework Galicia is used. For selection and refinement of components they wrote algorithms. This approach is limited up to component identification and connector identification is not considered.

Brian S. Mitchell and Spiros Mancoridis, *"On the evaluation of the bunch search-based software modularization algorithm"*, 2008. The Bunch algorithm extracted high

level architecture by clustering modules (files in C or class in C++ or Java) into sub-systems based on module dependencies. The clustering is done using heuristic-search algorithms. This approach first uses source code analysis tool to first create a graph of system structure, where the nodes are modules (e.g. Java classes/C++ files), and the edges are binary relations that represent the module level dependency (e.g. method calls, inheritance).The search based clustering algorithm has been implemented in Bunch tool. The tool generates a random solution from search space and then improves it by using evolutionary computation algorithms.

Sylvain Chardigny, Abdelhak Seriai, Mourad Oussalah, Dalila Tamzalit , *"Extraction of Component-Based Architecture From Object-Oriented Systems"*, 2008. In this paper proposed an approach called ROMANTIC which focuses on extracting a component-based architecture of an existing object-oriented system. It is a quasi-automatic process of architecture recovery based on semantic and structural characteristics of software architecture concepts. Software Architecture is extracted using a variant of the simulated annealing algorithm.

Sylvain Chardigny, Abdelhak Seriai, Dalila Tamzalit, Mourad Oussalah,*" Quality-Driven Extraction of a Component-based Architecture from an Object-Oriented System"*, 2008. It is quasi-automatic process of architecture recovery based on the quality characteristics of architecture by formulating it as a search-based problem. These characteristics guide the partitioning of the system classes in order to define architectural components.

The ROMANTIC tool uses metrics, but relies on a different approach than clustering. The first step of the extraction consists of defining a correspondence model between object concepts and architectural ones. This correspondence is elaborated by the architect. Then the tool validates this correspondence using predefined guides based on semantic and qualities of the architecture. The process selects among all the architectures that can be abstracted from a system, the best one according to the set of guides. The

guides are assumed to be measurable constraints to model the extraction process as a balancing problem of these competing constraints. The extraction problem is a search-based one and uses the Low-Temperature Simulated Annealing algorithm. The currently available information do not provide performance measures, the approach is costly, at least from a theoretical point of view. However, ROMANTIC is not yet publicly available to compare it with others.

Xinyu Wang,Xiaohu Yang,Jianling Sun and Zhengong Cai,"*A New Approach of Component Identification Based on Weighted Connectivity Strength Metrics"*, 2008. In this paper authors proposed component extraction method based on Weighted Connectivity Strength (WCS) metrics. The method proposed weighted connectivity strength metrics to measure connectivity between components and then applied clustering process to group classes based on WCS into components. On the basis of connectivity strength and considering variations of user defined types this study proposed new measure of component metrics WCS.WCS reflects the differences of user defined classes in the system and assign high weight to crucial classes, enlarge connectivity strength of classes related with crucial classes to help closely related classes easily cluster into component. The study also used hierarchical clustering algorithm for improvement of precision and efficiency. In this methodology interfaces between components are not identified.

Istvan Gergely Czibula and Gabriela¸Serban, *"Hierarchical Clustering for Software Systems Restructuring"*, 2007. In this paper author's proposed new agglomerative hierarchical clustering algorithm for restructuring of object oriented software systems in order to improve the structure of software system. For this purpose a heuristic that determines the no of application classes was proposed. This approach would help developers to identify appropriate refactoring. This approach consists of three steps: 1) Data collection-The existing software system is analyzed in order to extract from it the relevant entities like classes, methods, attributes, and the existing relationships between

them. 2) Grouping- The set of entities extracted in the previous step are regrouped into in clusters using hierarchical agglomerative clustering algorithm for improved structure of existing software system.3) Refactoring extraction- The newly obtained software structure is compared with the original software structure in order to provide a list of refactoring which transform the original structure into an improved one. The approach was evaluated on open source jHotDraw and results were obtained.

Onaiza Maqbool and Haroon A. Babri ,*" Hierarchical Clustering for Software Architecture Recovery"* , 2007.In this paper authors provided a review of hierarchical clustering techniques for architecture recovery and modularization of software systems, which is helpful for applying clustering successfully for the purpose of architecture recovery and modularization. As in the last few years, clustering has emerged as a promising technique for software architecture recovery. According to author understanding behavior of clustering measures and algorithm is the first step towards meaningfully employing clustering techniques for subsystem recovery. For this purpose: 1) they analyzed the behavior of various similarities and distance measures in the software context, thus identifying families of similarity/distance measures. 2) They analyzed the clustering approach of the Weighted Combined Algorithm (WCA) and LIMBO and described similarities between their two step approaches. The authors showed that these algorithms substantially reduce arbitrary clustering decisions that are common during the hierarchical clustering process in software domain. 3) They analyzed the clustering process of well-known hierarchical clustering algorithms and evaluated their strengths and weaknesses by using multiple assessment criteria. They demonstrated that the performance of an algorithm depends not only on its own characteristics but also on those of the software system to which it is applied. Thus the focus of this paper is on the analysis of hierarchical clustering measures and algorithms in the software domain and identification of their strengths and weaknesses in this domain so that they may be used effectively for architecture recovery.

Hironori Washizaki and Yoshiaki Fukazawa, *"A technique for automatic component extraction from object-oriented programs by refactoring",* 2005. In this paper authors concentrated on the extraction of components by refactoring Java programs. They proposed a technique for extracting components from existing object oriented programs by their new refactoring 'extract component' method. This extraction is based on the class relation graphs. Class relation graphs are obtained by static analysis of the dependencies among java classes. Then clustering algorithm was applied on graphs. In this approach authors first defined a class relation graph (CRG) that represents the relations among classes/interfaces in the target Java program. Next, using a CRG, they propose a technique for extracting components from OO programs, and changing the parts surrounding the extracted components to allow these surrounding parts to use the newly extracted components. These surrounding parts become the usage examples of the extracted components. This approach is limited to java beans components only.

Soo Ho Chang, Man Jib Han, and Soo Dong Kim, "*A Tool to Automate Component Clustering and Identification",* 2005. In this paper authors developed tool which identifies components from the object oriented system. The tool takes raw data input, which needs to be derived from fundamental artifacts of object oriented modeling such as use case model, object model and dynamic model. Hence, it is clear that if these artifacts are not available, it is difficult to identify components from the object oriented system. This means the method assumes that the fundamental artifacts of object oriented modeling such as use case model; object model and dynamic model are available. The approach consists of four steps:-1) measure functional dependency 2) clustering related use cases 3) allocate classes to components 4) Refine components. This method considers three types of relationships for identifying components. In step 1 and 2 functional dependency between use cases is used as the fundamental means to cluster related functions. The dependencies are measured with the four criteria in step 1 and related use cases are clustered in step 2. In step 3 functionality-to-data relationship expressed in dynamic model such as sequence diagram are taken to assign related classes to candidate

components. In step 4 dependency or coupling between classes is to verify and refine the identified components. If there are two closely related classes which are separated into two components, it is identified and refined in this step. Thus tool automates component clustering and identification method.

Eunjoo Lee Byungjeong Lee Woochang Shin Chisu Wu, *"A Reengineering Process for Migrating from an Object-oriented Legacy System to a Component-based System"*,2003 In this paper authors presented reengineering process for migrating from object oriented legacy system into component based system. The process consists of creating basic components using existing relationship and then refines the components by using metrics and clustering algorithm they have proposed. Components are retrieved from C++ source code. In this approach only dependency relationship among components is considered. The approach did not give much detail about the interfaces among the components. Lee et al defined criterions of component metrics, including connectivity strength, component complexity, etc. In the definition of connectivity, Lee assigned equal weight to all user defined types. However, the complexity of user defined types in a real system varies greatly, which is not reflected in Lee's definition and results in low precision of component classification. They created components based upon the original class relationships that they determine by examining the program source code. They described the system and process formally and suggested applicable metrics for the process. These can be used to help create components with the desired level of complexity that can operate as cohesive functional units in a distributed environment.

Woo-Jin Lee, Oh-Cheon Kwon, Min-Jung Kim, and Gyu-Sang Shin," *A Method and Tool for Identifying Domain Components Using Object Usage Information"*, 2003. In this paper authors presented a systematic method and its supporting tool called a component identifier that identifies software components by using object-oriented domain information, namely, use case models, domain object models, and sequence diagrams. These object oriented domain models were obtained from a domain analysis

process, in which common domain objects and common use cases were extracted through commonality and variability analysis. Assuming that common class diagrams, common use cases, and sequence diagrams are given after the domain analysis process, they focus on the component identification process, in which they clearly define dependencies among objects and propose object clustering algorithms. To precisely describe the dependencies among objects , authors merge the three viewpoints –structural, functional, behavioral into uniform model in which they extract the structural relationship among objects from class diagram. To clarify ambiguous dependencies among objects, they extracted object usage which represents usage relationship among objects such as create, destroy, update and reference, from sequence diagrams automatically or additionally specified the object usage according to use cases. The authors weighted each object usage according to the frequency or significance of each use case. To uniformly describe object usage and structural dependencies in a single notation they proposed an actor and object usage graph (AO usage graph). To perform the clustering algorithms they provide new graph concept called object dependency network. An object dependency network can be obtained from AO usage graph by calculating weighted value for the accumulated object usage and by eliminating actor nodes.. On the basis of object dependency network, authors provided two object clustering algorithms called seed algorithm and cohesion algorithm. In addition to this they provide supporting tool called object identifier.

Brian S. Mitchell, Spiros Mancoridis and Martin Traverso *," Search Based Reverse Engineering",* 2002. In this paper authors have described a process for reverse engineering the software architecture of a system directly from its source code, which consists of clustering the modules from the source code into abstract structures called subsystems and then reverse engineering the subsystem-level relations using a formal (and visual) architectural constraint language. This approach is especially helpful when other forms of traditional design documentation are outdated or not available. This approach consists of two steps supported by a suite of integrated tool. The first step uses their clustering tool, namely Bunch, to generate subsystem hierarchy automatically .

Using reverse engineered subsystem hierarchy as input , then they used a second tool , called ARIS(Architecture Relation Inference System) that enabled software developers to specify the rules and relations that govern how modules and subsystems can relate to each other. These formal descriptions are called interconnection styles and are created using visual architectural constraint language called ISF.

Hemant Jain, Naresh Chalimeda, Navin Ivaturi ,Balarama Reddy,*" Business Component Identification- A Formal Approach"*,2001. In this paper authors developed approach which helps in identifying components from analysis level object model representing a business domain. It is assumed that domain modeling has been done at analysis level which is input to the process .Thus domain model represents significant object classes using UML notations, the structural relationship between object classes, use cases and sequence/interaction diagrams presenting dynamic relationship between the classes. Author developed tool 'CompMaker' by implementing clustering algorithm for identifying initial set of components and then using super type, subtype relationship and set of heuristic enhance and refine the solution obtained from clustering algorithm. The approach uses Hierarchical Agglomerative clustering algorithm. For this approach, UML analysis model consisting of use case diagram, class diagram and sequence diagram needs to be prepared. User needs to have domain knowledge to assign weights to use cases.

Jong Kook Lee, Seung Jae Jung, Soo Dong Kim, Woo Hyun Jang, Dong Han Ham, *"Component Identification method with coupling and cohesion"*, 2001. In this paper authors proposed component identification method that considers class cohesion, class coupling, the quality metrics to define the quality of identified components. By using domain knowledge and experience of developer architecture design is performed.UML diagrams like use case, class diagram sequence diagram are used in it. Then component clustering algorithm is used to identify components using suggested component metrics. In this clustering algorithm mathematical basis is clustering binary relation, cluster

relation, class relation graph. For class relation graph developer's domain knowledge is required. Thus this approach consists of six steps:-1) Defining architecture 2) Design models using UML diagrams like use case diagram and class diagram. 3) Finding key classes 4) considering component cohesion 5) considering coupling among components 6) considering component interface. This approach is combined approach of clustering and graph.

Kamran Sartipi and Kostas Kontogiannis ,*" Component Clustering Based on Maximal Association"* , 2001. Authors presented a supervised clustering framework for recovering the architecture of a software system. The application of data mining techniques allows to extract the maximum association among the groups of entities. The user incorporates the knowledge about the system domain and documents into the clustering process. This approach first provides a new similarity metric based on maximal association property(maximum number of shared properties) between two groups of entities such as files. After this Supervised clustering technique is used for decomposing a large system of files into cohesive subsystems and finally used search space reduction technique to manage the search complexity. Authors implemented a prototype reverse engineering tool to recover the architecture of software system as cohesive components. Depending upon the user expertise and knowledge about the system, the user interaction can range from few steps of guidance to the clustering algorithm, up to determining a whole cluster. The tool represents the result of clustering as a subsystem and interconnections representation using both HTML pages to browse and analyze the quality of results and different graphs to visualize and investigate.

S. Mancoridis, B. S. Mitchell , Y. Chen, E. R. Gansner *,"Bunch: A Clustering Tool for the Recovery and Maintenance of Software System Structures"*, 1999. Mancoridis et. al proposed Bunch tool which can cluster source level modules and dependencies into subsystem. The tool assumes that the modules and dependencies of a system are mapped to a Module De-pendency Graph (MDG).The MDG is automatically constructed using

readily available source code analysis tools. The Bunch tool was extended to take into account human knowledge. The approach uses clustering algorithms to automatically partition software products into cohesive clusters that are loosely connected. Clustering algorithms, based on hill climbing and genetic algorithms are applied on module dependency graphs and extracted from source code.

Spiros Mancoridis and Brian S. Mitchell," Using automatic clustering to produce high-level system organizations of source codes, 1998. This paper describes automatic recovery of the modular structure of a software system from its source code. First step in this process is to extract module level dependencies from the source code and store resultant information in a database. Authors used AT&T's CIA tool and Acacia for C++ for this step. After all of the module-level dependencies have been stored in the database, they executed an AWK script to query the database, filter the query result and produce as output a textual representation of module dependency graph. The clustering tool Bunch is applied to their clustering algorithms to the module dependency graph. Then they used the AT&T's dotty visualization tool to read the output file from clustering tool and produce visualization of results.

Chung-Horng Lung," *Software Architecture Recovery and Restructuring through Clustering Techniques"*, 1998. In this paper author proposed a quantitative approach based on clustering techniques for software architecture restructuring, reengineering and recovery. Use cases are used together with the different clustering methods to reduce complexity at different levels of abstraction along with the design patterns. A visualization tool, SPV (Software Partition & Visualization) was developed on top of the clustering methods to provide a user friendly environment. Using two examples authors showed a result of decoupling effort of a legacy system and an application of the clustering technique to support the identification of a design pattern. This study also illustrates how the combination of use cases and clustering techniques help them restructure the system.

- **Graph Based or using Dominance Software Architecture Recovery**

Dominance analysis is a graph based technique to identify certain nodes in directed graph. The dominance analysis can be applied on call graphs derived from system to identify candidates for reusable modules and components in object oriented system.

A dominance is a relation between nodes in directed graphs G=(N,E), where N is a finite nonempty set of nodes and $E \subset N \times N$ is a set of edges. A root node of a directed graph is a node $r \in N$ with no incoming edges. A root directed graph Gr = (N, E, r) is a directed graph (N,E) with unique root node $r \in N$.

Thus in this approach mathematical graphs are developed either by static analysis or dynamic analysis whose nodes are classes and edges are interaction between classes. Using these graphs components are created.

Hassan Mathkour, Ameur Touir, Hind Hakami, Ghazy Assassa,*"On the transformation of object oriented-based systems to Component based Systems"*, 2008. The approach proposed a framework which creates component based software from object oriented based software.This approach consists of following steps:-1) Taking UML class diagrams as inputs; UML class diagram is generated of inputted java code and then exported to XMI. Open source tool ArgoUML is used for this purpose. 2) Analyzing the class diagrams to generate a graph; reading the class diagram's design elements and relations from XMI file which is output of the class diagram phase. Weighted directed graph is created for the XML file generated. Nodes of the graphs are elements such as classes and interfaces, the edges are relationship between those elements.3) Setting a weight for each edge of the graph according to the type of relation it represents. 4) Taking the weighted graph and clustering it into highly connected clusters; a hierarchical divisive clustering technique is used for clustered graph generation and based on graph components are created.5) Generating the result so that each cluster represents a

component.6) Producing fully deployable components using one of the available forward engineering tools.

Spiros Xanthos, *"Clustering Object-Oriented Software Systems using Spectral Graph Partitioning",* ACM Student Research Competition 2005.In this paper author proposed a method for analyzing object oriented software system trying to identify highly coupled communities of classes. Utilizing this he obtained the modules that form the system. Also this method can identify clusters that are autonomous and might possibly imply reusable components. Finally this method can estimate the degree of modularity in the software system by recognizing the individual modules that constitute the system. These are accomplished by applying Algebraic Graph theory techniques in the object oriented software domain. The innovation of this paper is use of spectral graph partitioning techniques in object oriented domain and the application of these techniques for decomposing an object oriented system into smaller modules, some of which might be used as reusable components. In this method author uses class diagram to create graph representation and then algorithm is applied to partition the graph into sub graphs. This is iterative process and the algorithm stops when external edges are more than internal edges. This methodology focuses on only component identification and not about the interface details among components created.

Spiros Xanthos,*" Identification of Reusable Components within an Object- oriented Software System using Algebraic Graph Theory"*, 2004. The approach for identifying reusable components from object oriented system has been developed. The technique used here is Spectral Graph partitioning. In this approach graph were created from class diagram in which classes stands for the nodes and the discrete messages exchanged between the classes stand for the edges. The approach is based on iterative method for partitioning graph in order to identify possible reusable components within system. From the graph eigenvectors of Laplacian matrix derived and is used for partitioning i.e.

algebraic graph theory is used for identifying reusable components. Thus class diagram needs to be generated and then spectral graph partitioning algorithm is applied.

Jonas Lundberg and Welf L¨owe , *"Architecture Recovery by Semi-Automatic Component Identification"*, 2003. In this paper authors proposed to use semi-automatic program analysis to extract the information. The overall process consists of starting point as input source code of the program about to be investigated. Then certain information as series of abstractions is extracted from source code. This information is used to construct a call graph, which is low level representation of program. Using this low level representation system architecture is recovered. The authors used dominance analysis to identify possible software components in an object oriented system. The actual dominance analysis is applied on a high level representation of the system i.e. the class graph – a directed graph where nodes are the system's classes and edges class interactions. It can easily be obtained from the system call graph. Dominance analysis was applied to class interaction graph, which was derived from object oriented system. In class graph, nodes are system's classes and edges are class interactions. Dominance analysis is good at identifying certain types of components but cannot be used to recover the complete architecture of the system at hand. Much more human intervention is required for component identification.

2.3 Other Approaches

Shaheda Akthar and Sk.MD.Rafi,*" Recovery of Software Architecture Using Partitioning Approach by Fiedler Vector and Clustering"*, 2010. In this paper authors proposed approach in which modules are identified i.e. procedures, files functions etc. Based on this information graph constructed and identified relations between modules. The input to graph are adjacency matrix, Degree matrix and Laplacian matrix. The graphs are decomposed into sub graphs using similarity measures. Finally clustering methods and the general notion of fielder vector are used for evaluating design patterns, which is part of Software Architecture recovery.

Shaheda Akthar, Sk.Md.Rafi, *"Improving The Software Architecture Through Fuzzy Clustering Technique", 2010*. In this paper authors used a fuzzy clustering technique to make the Software Architecture recovery to be more efficient and accurate. The approach uses one of the most popular clustering algorithm called the fuzzy C-means to find the related data items which share the common properties. The steps in this approach are : i) Identify the data sets present in the software. Ii) Calculate the degree of relatedness of these components iii) Apply the fuzzy C means algorithm to reconstruct the components obtained. This step involves with two phases- 1) Calculate the cluster centers 2) Assign these points to the clusters. This process is repeated until the cluster center is stabilized. Thus the architecture is recovered using fuzzy clustering.

Pascal Andr´e, Nicolas Anquetil, Gilles Ardourel, Jean-Claude Royer,*" Component types and communication channels recovery from Java source code"*, 2009. In this paper authors proposed tool which recognizes components, its type and communication channels in existing java source code. The approach explicitly identifies communication paths between existing components. This project aims at establishing link between component implementation that could be called the concrete model - and component specifications- that could be called the abstract model. The concrete model can be any object oriented application like java application. This research project tries to establish: 1) A common meta-model that addresses both the problem of handling several specific components models E.g. SOFA, Kmella etc. in a generic way and the problem of linking abstract models and concrete code. The meta-model also provides the data structure to store the traceability links between models and code and set of rules to check abstract models well-formed. 2) The structure abstraction tool extracts and infers architectural and typing features from source code. It is designed as an iterative and rule based process. 3) The behavioral abstraction tool extracts a specification of the dynamic behavior of the components identified during the structure abstraction process. It also works from static analysis of the source code. The input to project is java source code and output is the set of components with several kinds of relations between them and set of data types.

Abdelkrim Amirat and Mourad Oussalah , *"Enhanced Connectors to Support Hierarchical Dependencies in Software Architecture"*, 2008. In this paper authors proposed C3 (component, connector, configuration) meta model. Authors also proposed two complementary models to describe system's architecture. They used representation model to describe architectures based on C3 elements and reasoning model to understand , analyze the representation model. The core elements of the C3 representation models are components, connector and configurations, each of these elements have an interface to interact with its environments. The reasoning model is defined by four types of hierarchies and each type represents a specific views on C3 representation model different from others. The four hierarchies are: 1)The structural hierarchy used to show the different nested levels of system architecture. 2) The behavioral description hierarchy to show different level of system behavior, generally represented by protocols.3)The conceptual hierarchy to describe the libraries of element types corresponding to structural or behavioral elements at each level of architecture description. 4)The metamodeling hierarchy to locate where our model coming from and what we can do with it. Each hierarchy is associated with two points of view first the external view i.e. logical architecture. Second view is internal view i.e. physical architecture. The approach described software architectures which is a minimal and complete Architecture Description Language. They also introduced new concept of connectors.

Trevor Parsons, Adrian Mos, Mircea Trofin, Thomas Gschwind," *Extracting Interactions in Component-Based Systems"*, 2008. This paper covers dynamic techniques for collecting component interactions. It presented number of different approaches for capturing component level interactions. Authors presented approaches here cover the most widely used techniques for interaction extraction in enterprise Java systems. For each approach they presented need and technical requirement for implementation of approach. They used different tools to extract and recording interactions from java system. They also presented performance and functional consideration and contrast them against each other by outlining their relative advantage and disadvantages.

James Sasitorn and Robert Cartwright," *Deriving Components from Genericity"*, 2007. In this paper authors described how to formulate a general component system for a nominally typed object-oriented language supporting first-class generic types simply by adding appropriate annotations. The fundamental semantic building blocks for constructing, type checking and manipulating components are provided by the underlying first class generic type system. To demonstrate simplicity and utility of this approach to support components authors have designed and implemented an extension of Java called Component NEXTGEN (CGEN). CGEN is based on Sun Java 5.0 javac compiler backward compatible with existing code and runs on current Java Virtual Machines.

Stephen Kell ," *Rethinking Software Connectors",* 2007. In this paper author precisely characterized connectors, resolving many ambiguities and inconsistencies in the literature and contradicting the popular assumption that components and connectors are disjoint. The paper contributes : 1) A more precise characterization of connectors and relationship with coordinators and adapters. 2) They described the relationship between coupling and connectors and argued that connectors should be capable of adaption in order to maximize component reuse. 3) They identified the class configuration languages and stated their relevance to connections, proposing explicit configuration and suitable configuration language. Authors also described about what the connectors are and what aren't connectors.

Mircea Lungu and Michele Lanza, Tudor Gˆırba," *Package Patterns for Visual Architecture Recovery"*, 2006. In this article authors proposed a set of package patterns which are used for augmenting the exploring process with information about the worthiness of the various exploration paths. The patterns are defined based on the internal package structure and on relationships between the package and other packages in the system. Authors also proposed classification of packages based on information regarding the structural properties of the packages and on the way they interact with one another. When only the source code is available, recovering the architecture of a large software

system is a difficult task, authors presented this interactive visual approach to architecture recovery based on package information. This approach considers only dependencies among packages and automatically decompose the system based on package structure.

Ondrej Galik and Tomas Bures ," *Generating Connectors for Heterogeneous Deployment"*, 2005. Authors presented approach to create an extensible connector generator with features needed for heterogeneous deployment. They have designed an open framework allowing to add plug-ins for supporting different connector features (in the form of connector elements) and different component systems and their associated type-systems.

Authors have followed a connector model based on composing the overall connector functionality from small components (connector elements). They have designed an open framework allowing to add plug-ins for supporting different connector features (in the form of connector elements) and different component systems and their associated type-systems. We have implemented our approach in Java. The current implementation allows them to build connectors that comply with all the requirements brought in by the heterogeneous deployment.

Zhongjie Wang, Xiaofei Xu, and Dechen Zhan," *A Survey of Business Component Identification Methods and Related Techniques"*, 2005. Authors in this paper presented various component identification methods. Authors classified these methods into four types i.e. domain analysis based methods, cohesion coupling based clustering methods, CRUD matrix based methods and other methods. In domain engineering based methods, component designers do domain analysis from a group of similar requirements in one business domain, find commonalities and variables across them, construct domain specific software architecture to seek reusable business semantics, then construct reusable business component specifications. As any software artifacts require changing itself along with time these methods are not reasonable. Basic idea of Cohesion coupling based clustering methods are : calculate the strength of semantics dependencies between two

business elements and transform business model into the form of weighted directional graph, in which business elements are nodes and semantics dependency strength are the weight of edges between nodes, then cluster the graph using graph clustering or matrix analysis techniques. CRUD matrix based methods are actually a clustering method which uses those behavioral business elements(e.g. use case, events, operations) and static business elements(e.g. business entities) as sample data, uses four semantic relationships(Create-C, Read-R, Update-U, Delete- D, with priorities as C>D>U>R) between behavioral and static elements to calculate association weight and merges those use cases and entities with C or D relationships into one business components. Other methods include Similarity based component identification method, Variation Oriented Decomposition Method, Information loss Minimization based method, Business Model stability based method etc. These methods lacked complete methodology for component and connector identification. More over these methods have less automation degree.

Andrey A.Terekhov,*" Dealing with Architectural Issues: a Case Study"*, 2004. In this paper author tried to recover and improve software architecture in a large-scale industrial project. Author presented a case study in software architecture recovery and transformation.

Smeda, A., Oussalah, M., and Khammaci, T, *"Improving Component-Based Software Architecture by Separating Computations from Interactions"*, 2004. In this paper authors presented approach in which authors justify why connectors should be separated from components and treated as first-class entities, while describing component based Architecture, As most of the ADL (Architecture description language) defines connectors implicitly. The approach used by author is known as COSA (Component based Software Architecture) in which connectors are defined explicitly by separating their interfaces from their implementations and configurations. COSA connector is mainly represented by an interface and a glue specification. The interface shows the necessary information about connector, including number of roles, service type that the connector

provides(communication, conversion, coordination, facilitation),connection modes(synchronous, asynchronous),transfer mode etc. The glue specification describes the functionality that is expected from connector. It could be simple protocol links the roles or it could be a complex protocol. In short glue of connector represents the connection type of that connector. Therefore different deployment of components and connectors can be obtained resulting in different architectures of the same system.

Vijayan Sugumaran, Veda C. Storey,*" A Semantic-Based Approach to Component Retrieval",* 2003. In this paper authors developed semantic-based approach to component retrieval. A reuse repository was developed that contains the components relevant for the creation of new applications, along with their attributes and methods that uses Web and JavaBeans technologies. Authors developed component retrieval approach which consists of creating: 1)a reuse repository of design objects or components; 2)a domain model that contains meta level knowledge about the reusable components presented in terms of objectives, processes, actions, actors and objects; 3)an ontology that supports an interpretation of the meaning and use of application domain terms(for both the reusable repository and the domain model); and 4)a natural language interface for expressing queries .Thus initial query generation, query refinement, component retrieval and feedback through above steps is performed. In this approach user executes query for component retrieval.

Bridget Spitznagel and David Garlan, *"A Compositional Approach for Constructing Connectors",* 2001. Bridget et.al introduced an approach to connector construction based on incremental transformation. Authors defined connectors as a six tuple-[c,l,s,t,p,w}, where c is application level code that appears within component or compilation unit, l is – communication libraries , generated stubs etc, below application level , s is low level infrastructure services provided by operating system. t- is data/ tables . p- is a policy documenting the proper use of these parts and w is formal specification describing the connector's proper behavior. A connector transformation modifies one or more parts of

an existing connectors and resulting into new connector. Using set of different transformation new connectors can be constructed. For this authors have developed prototype tool which transfers Java RMI based i.e. basic interactions into new type of connectors. Basically this work is viewed as a step towards a more comprehensive engineering basis for component integration. They need to demonstrate for other kind of interactions beyond RMI.

Hassan Gomaa, Daniel A. Menascé and Michael E. Shin , *"Reusable Component Interconnection Patterns for Distributed Software Architectures"*, 2001. Hassan Gomaa et. al have described the design of reusable component interconnection patterns in client/server systems. Pattern which define and encapsulate the way client and server components communicate with each other using UML. Given these patterns, the designer of a new distributed application can select and reuse the appropriate component interaction patterns. So, this method is for selecting components and interfaces which are already created.

Young Ran Yu, Soo Dong Kim ,Dong Kwan Kim, *"Connector Modeling Method for Component Extraction"*,1999 In this paper authors proposed a method that extracts domain specific components for a particular business domain using the connector model. Requirement specification was used for Use case model, class diagram and connector extraction. This approach consists of three phases:- connector modeling, component modeling and implementation. Connector modeling phase consists of use of requirement specification for connector extraction and for use case & class diagram modeling and proposed new diagram i.e. requirement diagram. Using class diagram ,use case and connectors components are extracted. In component modeling phase connector specification is modified to find interfaces of components. And components extracted in previous step are specified in more detail. Finally software Architecture is described in particular format. Thus, connectors are used as tools for extracting components instead of connector as interactions.

Helgo M. Ohlenbusch and George T. Heineman, "*Composition and interfaces within software architecture"*, 1998. In this paper authors explored the part that composition and inheritance play in defining interfaces using ports and roles. Author discusses these concepts within the context of the JavaBeans component model and shows how to capture the complexity inherent in the interfaces of components and connectors.

Ivan T. Bowman and Richard C. Holt," *Software architecture recovery using Conway's law"*, 1998. In this paper authors have introduced the idea of ownership architecture for a software system, and have shown how such a structure is useful in reverse engineering. It is a useful mechanism for predicting system structure. Authors presented three case studies using Conway's law. For each of the systems as case study authors presented following architectures:1) A conceptual architecture based on available system documentation. 2)An ownership architecture extracted from system documentation or revision control log.3) A concrete architecture extracted from the actual system implementation.

Robert Allen and David Garlan , "*A Formal Basis for Architectural Connection*" , 1997. Robert Allen and David Garlan presented a formal approach to one aspect of architectural design: the interactions among components. The key idea is to define architectural connectors as explicit semantic entities. In this approach connectors are treated as types that have separable semantic definitions (i.e. independent of component interfaces), together with the notion of connector instantiation. Authors used theory of algebras to show connector specifications.

Robert Allen and David Garlan," *Formalizing Architectural Connection"*, 1994. Robert Allen et. al. presented a theory for the interactions between components, which shows the most important aspect of architectural description . The key idea here is to define architectural connectors as explicit semantic entities. Authors provide a formal basis for specifying the interactions between architectural components by describing and reasoning about architectural connection also by assuming certain component types and

connector types. The description of these connector types is based on the idea of adapting communications protocols to the description of component interactions in software architecture. This approach provides notation and underlying theory for architectural connection explicit semantic status.

Summarizing the above papers, it can be said that software architecture recovery of object oriented system is an active and important area of software engineering research.

2.4 Observations from Literature Review

- Quasi - manual techniques are systematic and good and also able to extract components but requires lot of time and human efforts.
- Few of the good semi-automatic techniques are available which requires help of other tool and partially automated.
- Large numbers of quasi-automatic techniques are available in the literature. Many algorithms are proposed and implemented in tools. Mathematical concepts like graph theory and data mining algorithms are used. but reverse engineer has to drive the process.
- Most of previous studies focused on quasi automatic and applying clustering algorithms.
- Most of quasi-automatic approaches requires more inputs other than source code , which may not be available all the time for legacy system and then it need to be generated by any way first.
- Few of the approaches combines clustering and graph approach together for software architecture recovery.

2.5 Limitations of Existing Methods

Software architecture recovery approaches discussed above have following draw backs.

- Quasi manual approaches by S.K.Mishra[74] and Suk Kyung [91] discussed above identified components properly but most of the task needs to manually, which is time consuming. More over only components are identified, no guideline for connector identifications.
- Nenad's [55] approach identifies components and connectors but approach is manual and time consuming.
- For some manual approaches like Suk Kyung's approach [91] design specification of object oriented system should be available for migration into component based system which is not possible for every legacy system.
- Some quasi-automatic approaches like Soo Chang et al approach [86] requires fundamental artifacts of object oriented modeling such as use case model, object model and dynamic models available for component identification which may not be available for object oriented legacy systems. For such systems the approach is not suitable. This method also does not consider about the interface details among components.
- Quasi automatic graph based approach proposed by Hassan Mathkour [29] also identifies components and no interface details are provided.
- Some of the approaches like Simon Allier's approach [84] are limited up to component identification and connector identification is not considered.
- In some Quasi-automatic approach like Woo-Jin's approach [102], common class diagrams, common use cases, and sequence diagrams need to be given after the domain analysis process, they focus on the component identification process.
- For some approach like Hemant Jain's approach [32], UML analysis model consisting of use case diagram, class diagram and sequence diagram needs to be prepared. User needs to have domain knowledge to assign weights to use cases.
- For the approach used by Jonas [44] Much more human intervention is required for component identification . Moreover interactions among components is not recovered.

- Some of the approaches requires architectural style, conceptual architecture, architectural properties etc as input other than source code, which may not be available and then need to be generated by any means first then the recovery process starts.

It is also evident from the review of literature on software architecture recovery techniques that even though the domains and techniques of recovering architecture have varied with time, to the best of our knowledge, none of these gives automatic component retrieval only with the help of source code as input to the tool . Since industry is migrating from object oriented system to component based system as components more reusable and beneficial than objects. It is important to have a tool which help software developer or software maintenance person to create components and also interface among these components. So this study aims at finding such method and tool to recover components and interface details

2.6 The Present Study

The present study has an objective to propose quasi- automatic methodology and develop tool that will display components that can be created and interface details automatically.

In the present study approach is to reduce time efforts of software developer or maintenance person for migrating into component based system. Thus, reusing existing code and migrating to new environment saves cost, efforts of redesign and redeveloping the system which suits to new evolving environment. This is what the software industry always prefers.

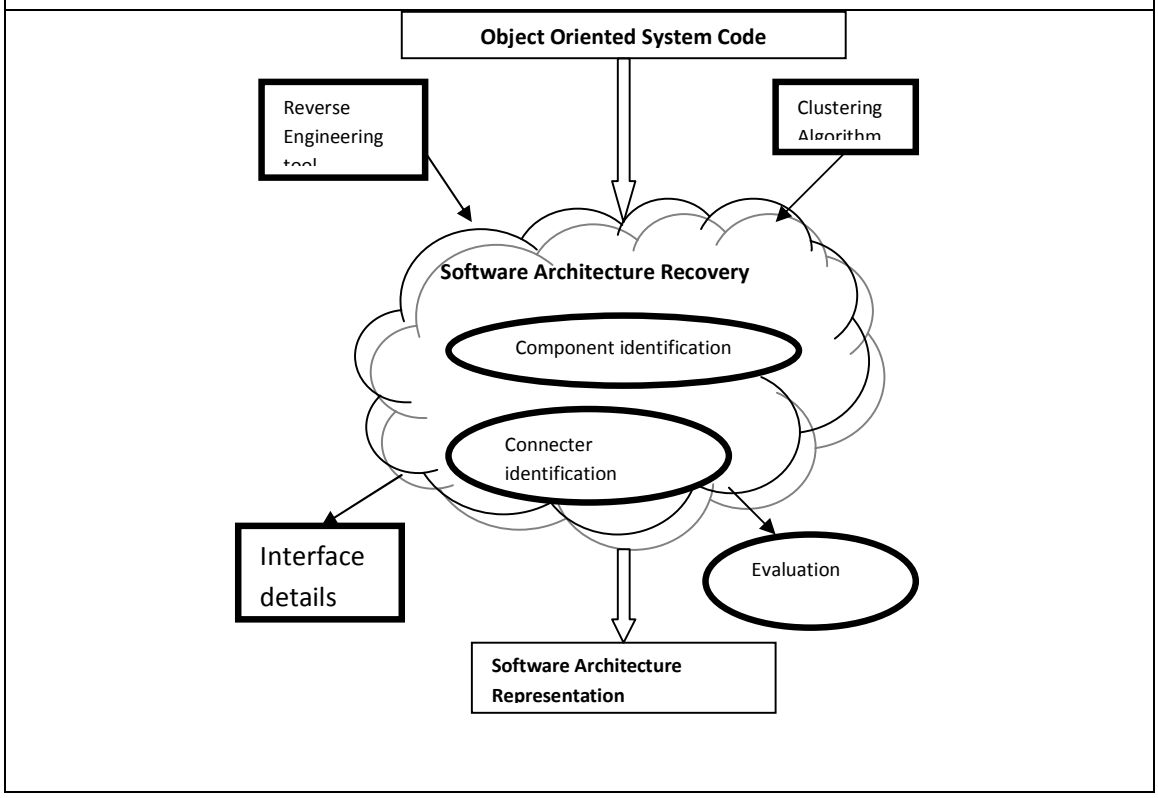
Thus the study proposes agglomerative clustering algorithm for creating components from object oriented system and implement it into proposed tool. The study will also focus on using and proposing Component Cohesion Metrics(CCM) for component evaluation. The interface details will also be extracted so that connector classes can be

created for the components. In the present we will focus on object oriented system developed in java.

Proposed Model:

The Figure 2.1 shows the entire proposed approach. The approach uses java source code as input to the model. It recovers software architecture by using clustering algorithm. The approach takes help of existing reverse engineering tools to verify all the classes from source code is covered or not. The approach recovers components and interfaces i.e. connector as a part of software architecture recovery. The approach also evaluates components for quality using metrics. Thus software architecture representation is ready.

Figure 2.1: Proposed Approach



Chapter 3

Study of Existing Reverse Engineering Tools, Framework and Selecting Clustering Process for Proposed Methodology

3.1 Study of Existing Reverse Engineering Tools

Software Engineering research and industry recognize the need for practical tools to support reverse engineering activities. Most of the well-known CASE-tools now a day's support reverse engineering in some way or other. Reverse engineering is first step towards software Architecture recovery. The most commonly used standard today is Unified Modeling Language to depict the architecture and design of an application. An UML class diagram describes the architecture of object oriented programs. Class diagram captures the essence of its design.

3.1.1 Extracting Classes from Given Object Oriented System using Tool

As proposed approach focuses on software architecture recovery when design document of legacy object oriented system is not available. We need to extract class diagram of legacy application to cross verify with our proposed tool results whether all the objects from the application system are considered in component creation or not. Class diagram shows classes and relation between them and it is necessary for creating components, as it will help in reconstructing the software architecture from existing implemented software. The idea behind choosing these tools was assessing different kinds of tools like commercial, non-commercial, open source tools that support reverse engineering of java

code. For this we have assessed capabilities of software reverse engineering tools to generate class diagram from java source code. Proposed framework is designed and implemented assuming that no design documents of legacy object oriented system is available. Hence we need to retrieve static structure of the object oriented system. UML class diagram provides this structure in the form of different classes in the system and relationship between the classes. Different reverse engineering tools are available in the market which take input as source code and give output as class diagram. Four tools were selected in this study as they support java reverse engineering. These are IBM Rational Rose, ArgoUML, Reverse, and Enterprise Architecture (EA).

Rational rose - Rational Rose is a widely used commercial UML modeling tool. Rational Rose offers reverse engineering capabilities, but their capabilities are very limited. Rational Rose supports reverse engineering of Java software systems. When reverse engineering a Java program, Rose constructs a tree view that contains classes, interfaces, and association found at the highest level. Methods, variables etc. are nested under the owner classes. Rose also constructs (on demand) a class diagram representation of the extracted information and generates a default layout for it. Additionally, Rose automatically constructs a package hierarchy as a tree view. Rose is able to reverse engineer the information from the source code (.java files), byte code (.class files), jar files, or packed zip files. In Rose, the Java reverse engineering module can be given instructions on files, directories, packages, and libraries to be examined.

ArgoUML - ArgoUML is a widely used open source tool for UML modeling tool. ArgoUML provides a modular reverse engineering framework. Currently Java source code is provided by default and there are modules for Java Jar and class file import. Similar to Rose ArgoUML constructs a tree view that contains classes, interfaces and association found at the highest level. Methods, variables etc. are nested under the owner classes. Using Drag and drop facility user can create class diagram. Reverse engineering capability of the tool is very limited as it cannot extract association and interface.

Reverse - Reverse is non-commercial tool to convert java code to class diagram developed by Neil Johan. User needs to select main java file and tool automatically displays class diagram. Tool has extracted limited classes, but no interfaces. Hence, realization relationships have not been extracted. It was successful in identifying most of the associations.

Enterprise Architecture (EA) - Enterprise Architecture (EA) is widely used commercial UML modeling tool. Tool generates tree view of classes and methods. Variables are nested under methods. EA's current reverse engineering capabilities can only reverse engineer UML semantics such as class diagrams and associations.

To assess the capability of these tools we examine following model properties of the tools-

3.1.2 Examine Model Properties of these Tools

Number of Classes (NOC) -This is a general measure for the overall size of a software module. Therefore, high NOC values may indicate a more detailed representation.

Number of Associations (NOA) - NOA is a metric measure of interconnectedness in a module. In reverse engineering it is important to understand how classes are connected.

Number of Generalization relationship (NGR) – It models "is a" and "is like" relationships, enabling you to reuse existing data and code easily. It is a generalization / specialization relationship between classes, which helps to measure how tightly coupled classes are. From reverse engineering point of view it will help for concluding component structure.

Handling of Interfaces - An interface is a specifier for the externally-visible operations of a class, component, or other classifier (including subsystems) without specification of internal structure. In UML diagrams, interfaces are drawn as classifier rectangles (with a stereotype << Interface >>) or as circles. The interfaces are attached by a dashed generalization arrow to classifiers that support it, known as realization relationship. This indicates that the class provides (implements) all of the operations of the interface. The circle notation is used when the operations of the interface are hidden .A class that uses or requires the operations supplied by the interface may be attached to the circle by a dashed arrow pointing to the circle. From the reverse engineering point of view, generation of such dependencies is important for understanding the usage of interfaces and for concluding component structures and dependencies (e.g., to abstract class diagrams to a component diagram). Furthermore, different ways of handling interfaces have impact on the NOC metric and possibly on the readability of the respective class diagram.

Role Names - The function of role names at association ends is comparable to that of attribute names in the sense of giving to an association between classes a meaningful descriptor, which depends on the end it's attached to. Therefore in reverse engineering, role names can hold relevant additional information about the system infrastructure. We examine, whether role names are used and if, what kind of information they represent.

These model properties are beneficial to create components and connectors in component based architecture. Hence dependencies generated through our tool are verified with class diagram generated through one of the above tool (i.e. Enterprise Architecture). As, we found that Enterprise Architecture tool was able to extract maximum information from the object oriented source code like all the classes ,interfaces, basic relationship among the classes like inheritance, composition etc.

Thus, dependencies generated through the proposed tool are verified with class diagram generated through one of the above tool (i.e. Enterprise Architecture). We found that Enterprise Architecture tool was able to extract maximum information from the object oriented source code like all the classes, interfaces, basic relationship among the classes like inheritance, composition etc.

Steps to Examine Tools to Generate Class Diagram:

- Keep input source code in one folder.
- Open tool and do reverse engineering.
- Classes along with attributes and method will be extracted.
- By dragging classes into framework of tool, class diagram will be prepared.
- Count manually number of classes retrieved (NOC), Number of association relationship retrieved (NOA), Number of generalization relationship retrieved, and roll names retrieved.
- Do the comparison of tools based upon the counts.
- Decide which tool extracts all the information.
- Use class diagram generated from the tool, which extract maximum information above, to compare results from module one of our tool.

3.1.3. Comparison of the Tools

For the comparison of reverse engineering tools we chose following application as

Case study: We have chosen small java software. ‘Arithmetic24 Game’. This is a software game application developed in Java by Huahai Yang. It is a simulation of popular traditional card game. It consists of 19 classes and 1 interface.

Following are results from different tools. The static elements found by the CASE tools are classes, association relationship, generalization relationship, interfaces and roll

names. Comparison of the elements found by these tools is shown in table 3.1 and Class diagrams generated by the tools are shown in Figure 3.1 –3.4.

Classes–IBM Rational Rose, ArgoUML, and Enterprise Architecture were able to find 19 classes, when applied to the "*.java" files ofArithmetic24.Tool Reverse was able to find out only 17 classes as this tool accepts only main java file for reverse engineering. The name compartment of the class reverse engineered by all the four tools contains the name of the actual
class. Both the attributes and operations compartments contain the names, types and visibility (public, private or protected) for Rose, EA, and Reverse. ArgoUML could not identify visibility.

Associations - Total number of associations found by Rose and EA was 12.ArgoUML could not find any association. Reverse found 18 associations and showed mutually dependent classes with red dashed line. Associations are directed in all cases except for ArgoUML. The roll is named by the variable itself. Rose and EA could produce roll names. For tool 'Reverse', associations are directional but do not specify any roles.

Generalization - All the four tools were able to recognize generalization relationships. All the tools found a total of 6 such relationships.

Handling of Interfaces - Rose uses a circle to illustrate interfaces in the class diagram. The (abstract) methods of the interfaces are written below the circle, separated with two horizontal lines, which is not recommended in UML.EA illustrates them using class rectangles with a <<Interface>>stereotype shown above the interface name. This notation is also available as an option in Rose. Both Rose and EA found one interface in the Arithmetic24 core package. Both connect the interfaces to the classes that support them, that is, the classes that implement the abstract methods defined in the interfaces. Rose does that with a solid line (a realization relationship). EA uses a dashed line with a

triangle at the end pointing to the interface (similar to the inheritance notation).However, neither Rose nor EA were able to generate any dependencies between interfaces and the classes that use them (typically shown with a dashed arrow from a class pointing to the interface). This is an obvious limitation to understanding the roles of the interfaces. Further, interface dependencies are needed for abstracting a class diagram into a component diagram, understanding the interaction among different components, etc. Tools ‘Reverse’ and ‘ArgoUML’ could not able to extract interface.

Figure 3.1 : Class Diagram of Arithmetic24 game from Rose

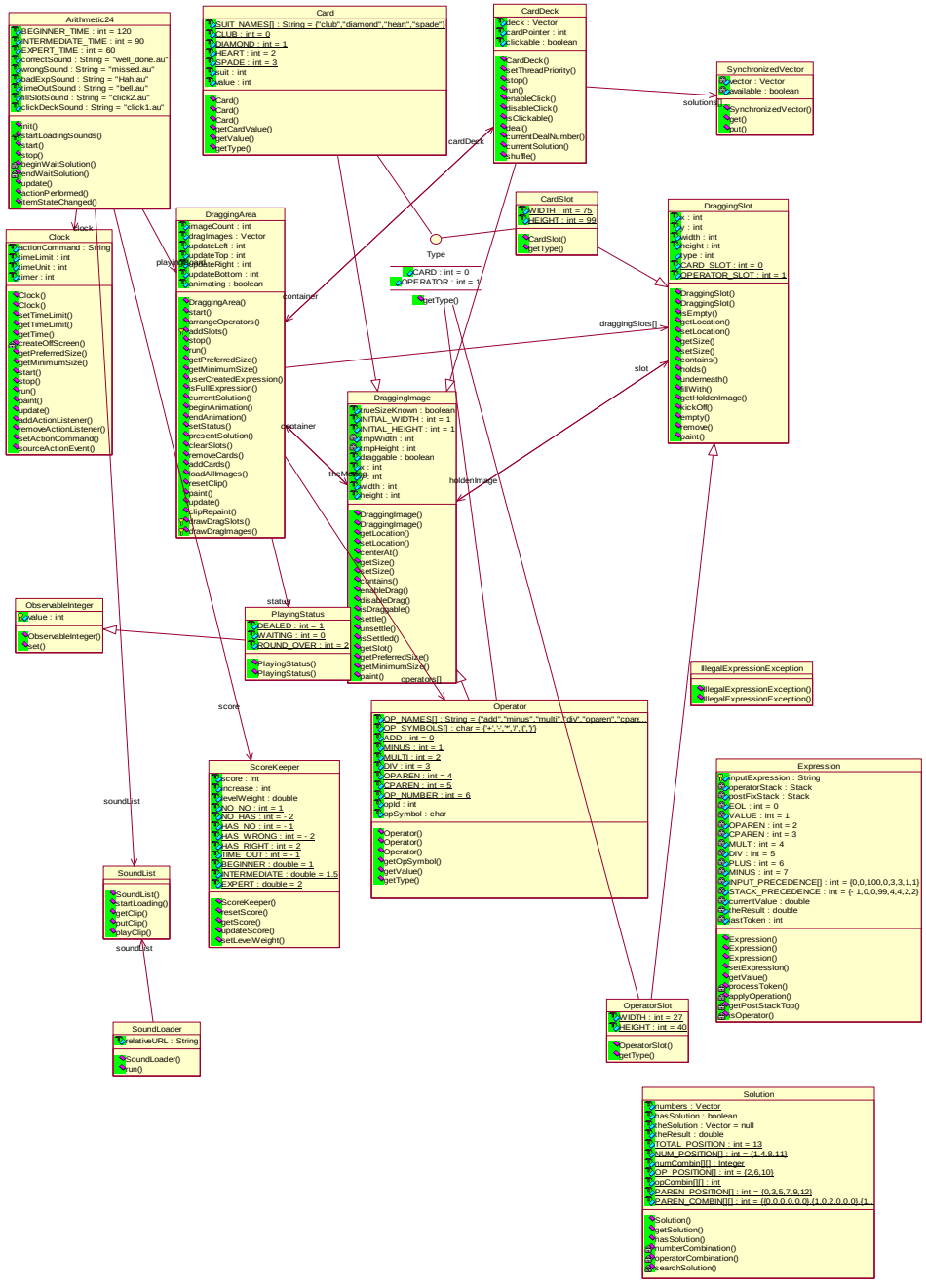


Figure 3.2: Class Diagram of Arithmetic24 game from ArgoUML

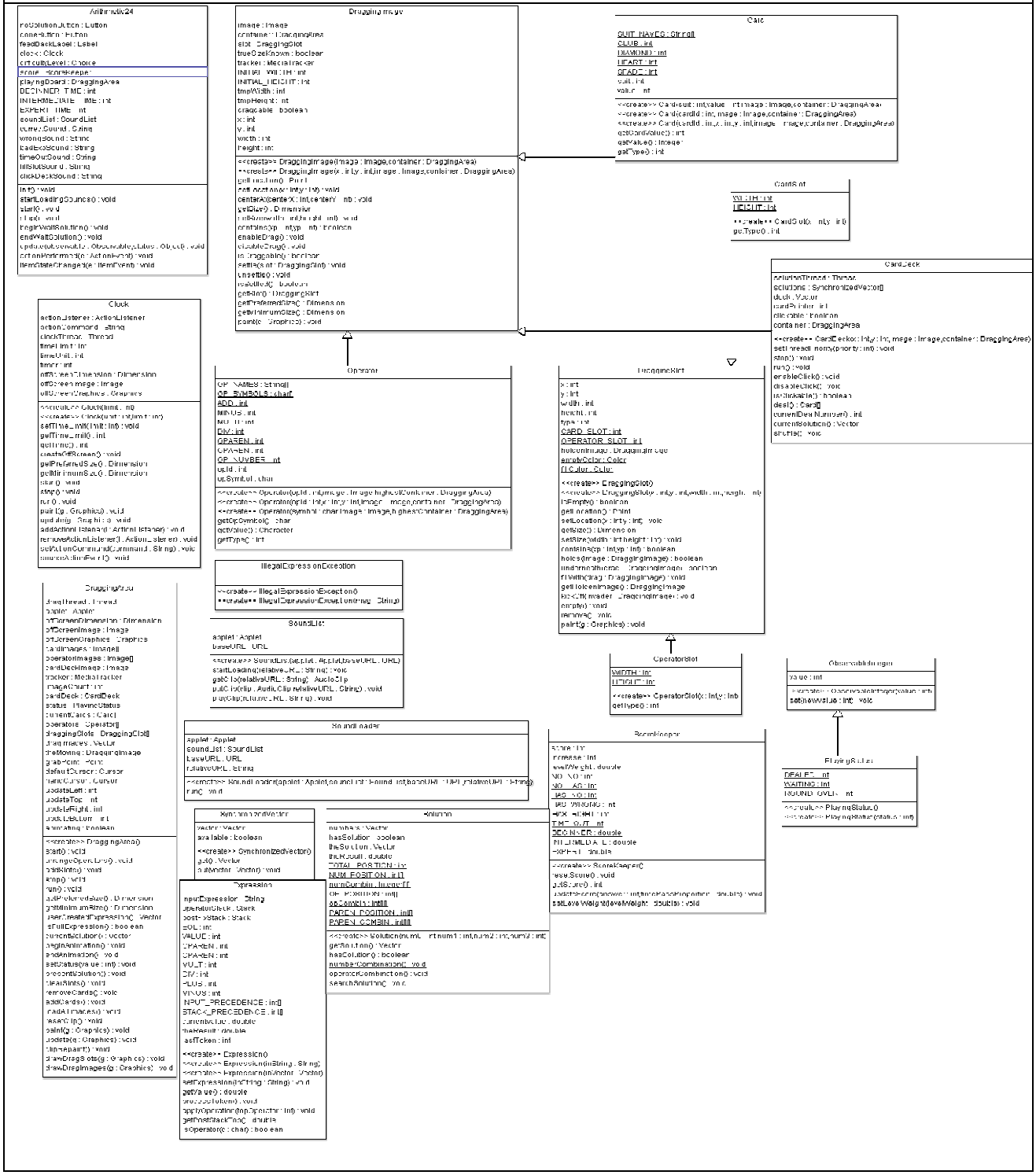


Figure 3.3: Class Diagram of Arithmetic24 game from Reverse

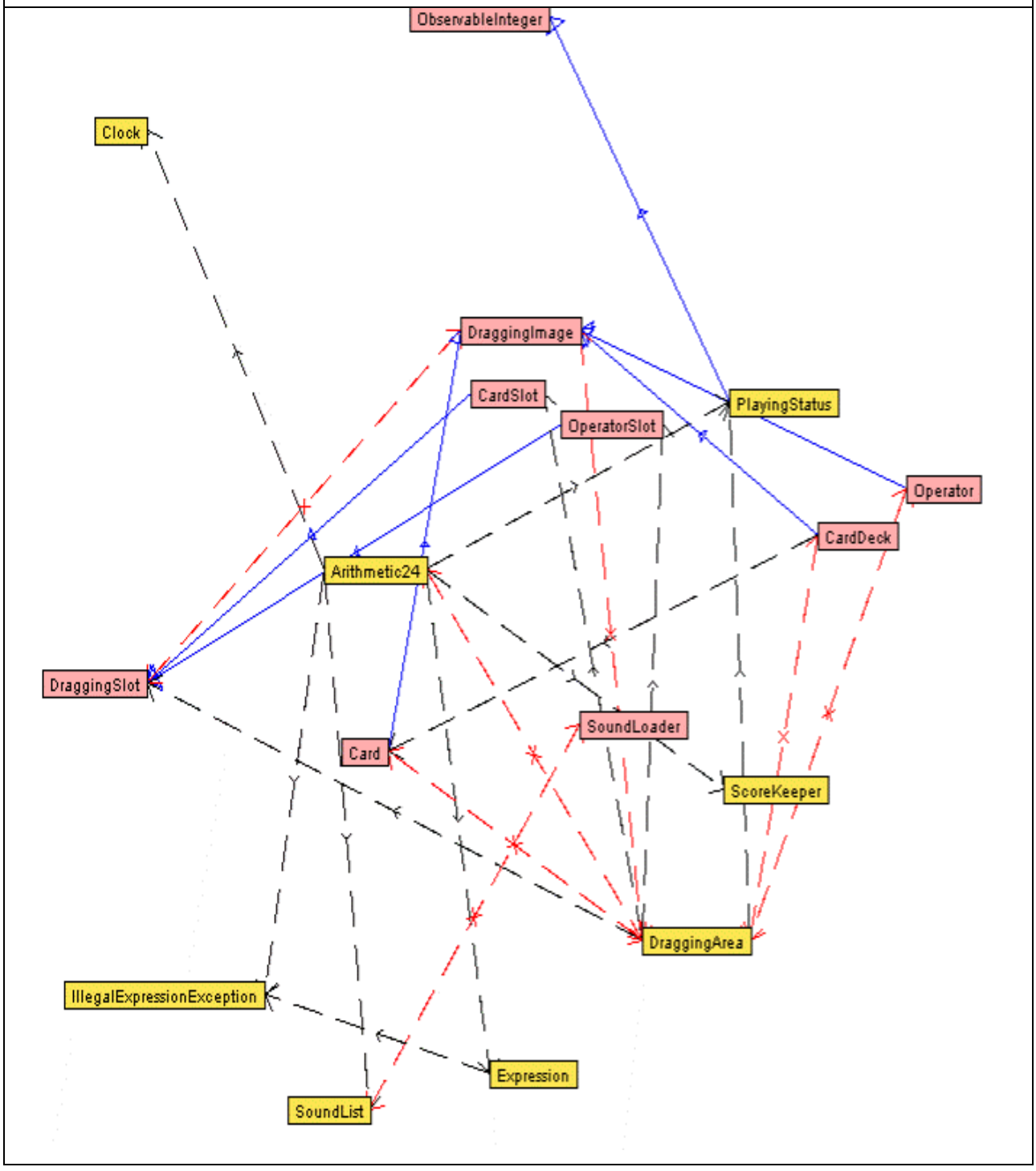
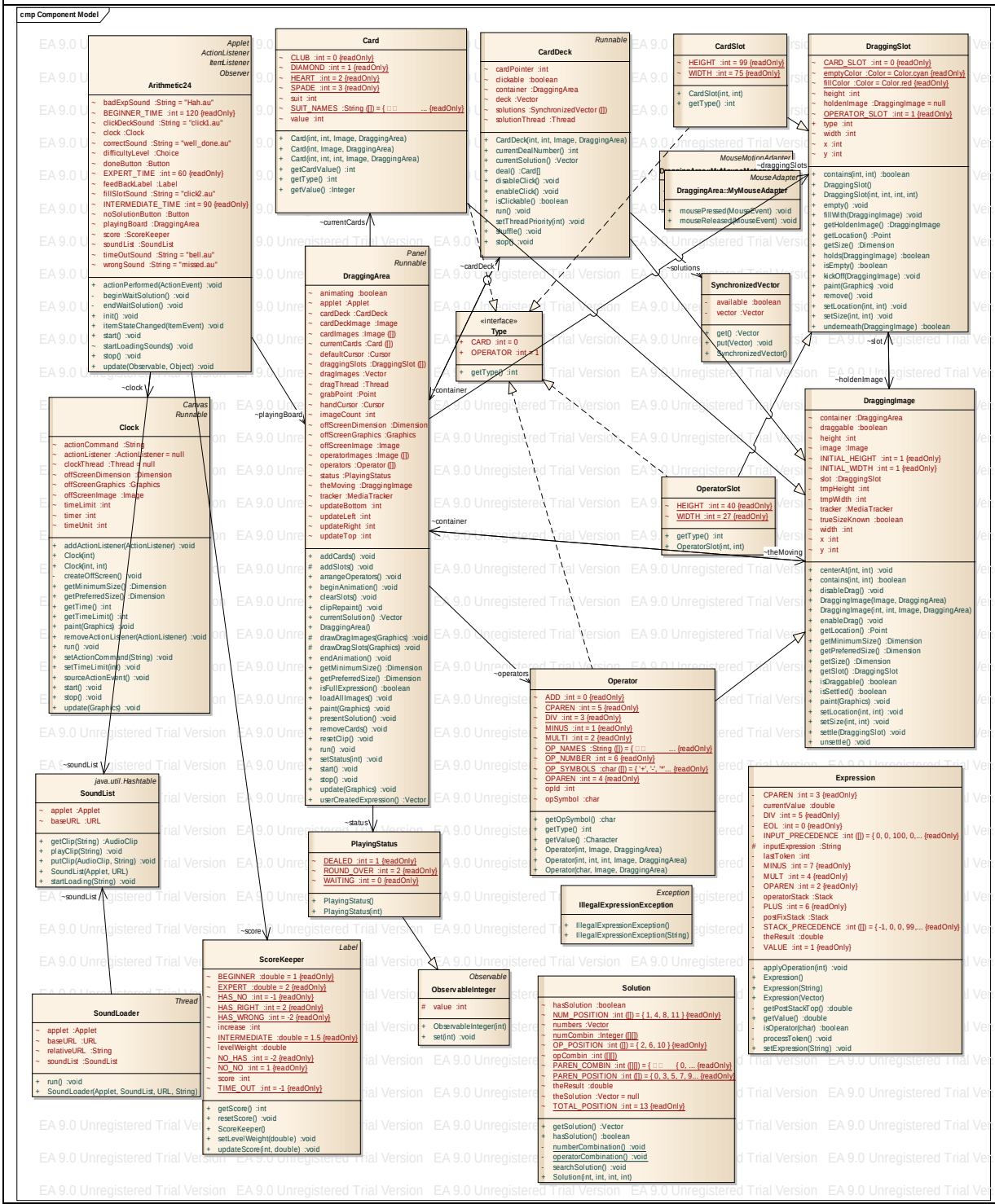


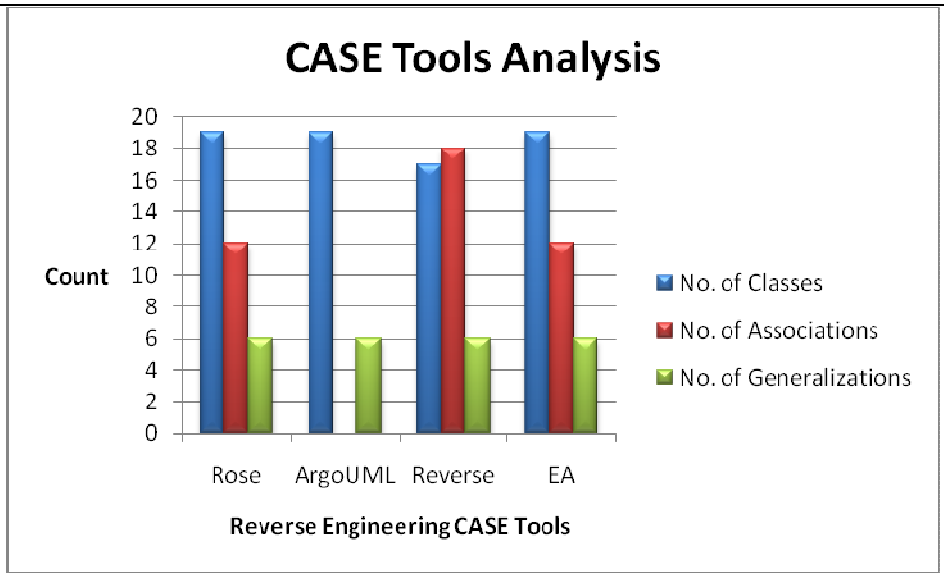
Figure 3.4: Class Diagram of Arithmetic24 game from Enterprise Architecture



Tools	No. of Classes	No. of Associations	No. of Roll Name	No. of Interfaces	No. of Generalizations	No. of realizations
Rose	19	12	14	1	6	4
ArgoUML	19	0	0	0	6	0
Reverse	17	18	0	0	6	0
EA	19	12	15	1	6	4

Table 3.1: Elements found by CASE Tools

Figure 3.5: CASE Tools Analysis Chart



Analysis:

In the present study we have assessed capabilities of four reverse engineering software tools that generate class diagram from java source code. We have found that, most of the classes are identified with simpler relationships. In the present study, four tools have been compared with regards to their reverse engineering capabilities. We have carried out

manual comparisons. The manual comparison is needed to understand the interpretations and mappings used to generate a class diagram.

We observed that most of the classes are extracted by all the four tools but all the relationships have not been extracted properly. The simpler inheritance, associations and realization relationships were extracted. Few of the classes remained unrelated to any of the classes in the diagram; even if source code shows the classes are related. ArgoUML and Reverse were unable to extract interfaces and realization relationships.

All the above tools, except 'Reverse' need to drag and drop the classes to complete class diagram, once reverse engineering is complete. Reverse automatically generates the class diagram but all classes are not extracted. We conclude that Rational rose and Enterprise Architecture extracts maximum required static information. So any one available tool can be used to generate class diagram.

3.2 Study of Existing OSGi Framework for Implementing Components Created

Once the object oriented application is restructured into a component-based application, we need to reorganize it according to a concrete component model to make it operational. To illustrate this, we choose to use the OSGi component model.

3.2.1 OSGi Model:

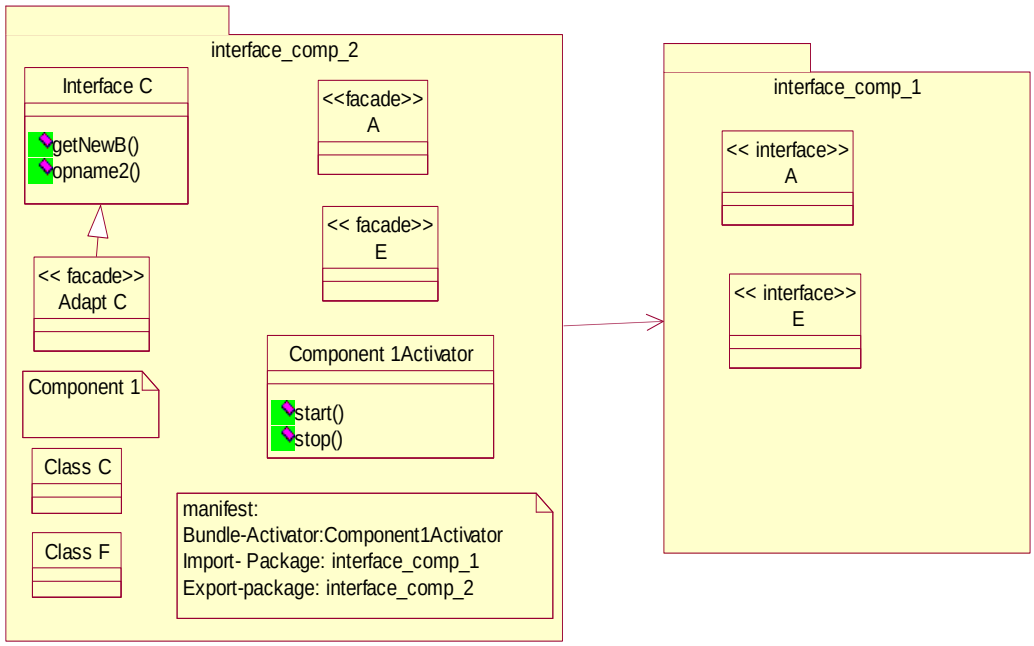
Andre L. C [6] presented introduction of OSGi, the Open Services Gateway Initiative (OSGi) is a framework that supports the implementation of component-based, service-oriented applications in Java. The framework manages the life-cycle of modules (called bundles in OSGi) and provides means to publish and search for services. It supports the dynamic install and uninstall of bundles. Nowadays, OSGi is used in many application domains, including mobile phones, embedded devices, and application servers. Basically, bundles are regular Java JAR files containing class files, other resources (images, icons,

required APIs etc), and also a manifest, which is used to declare static information about the bundle, such as the packages the bundle import and export. Furthermore, bundles may provide services to other bundles. In the OSGi architecture, a service is a standard Java object that is registered using one or more interface types and properties (that are used to locate the service). Another key component of the OSGi run-time is the Service Registry, which keeps track of the services registered within the framework. Following section provides guideline for bundle creation.

3.2.2 Creating Bundle using OSGI Framework: In the OSGi framework, a component (called bundle) is a set of classes organized into packages, which are by default not visible to outside the bundle. With the help of manifest it is possible to export packages. Classes and interfaces in these exported packages become visible to outside bundle. Thus they act as provided interface. Similarly it is possible to indicate packages that the component requires to operate. Consequently, classes and interfaces of these packages play the role of required and provided interfaces.

In order to export the provided interfaces of our components, through the manifest, we place them in specific packages. Similarly the required interfaces are specified in manifest by importing the packages containing them. Indeed, then these are necessarily exported by other components.

Figure 3.6: Example of OSGi Bundle



For example, suppose in figure3.5, interface_comp_2 is bundle1, which is package interface_comp_2 contains provided interface, *Interface C* and bundle 2 i.e. interface_comp_1 contains required interfaces, *Interface A* & *interface E*. All this is specified in the manifest as follows:

Import package: interface_comp_1

Export package: interface_comp_2

3.2.3 Activators Management in OSGi Framework

Once the object oriented application is restructured accruing to the concrete component model, its launch must conform to the framework of this model. The OSGi framework allows the specification of actions to be performed during the different phases of bundle's lifecycle using the class *BundleActivator*. Thus, using this mechanism to launch

restructured application, for each class containing an entry point (i.e. main () method in Java), we create in its corresponding bundle a subclass of the class *BundleActivator*that redefines the method *start (BundleContext).* These subclasses are potential activators of the bundle. The redefined method is only used to call the original entry point (i.e. main () method in Java) of the application. Its parameter *(BundleContext)* contains, among others, the parameter of the main () method. Among all the potential activators of the bundle, the designer should designate the actual one. It is identified in the manifest as follows:
BundleActivator:

Activator.Component1Activator

Finally, to build an OSGi bundle, the classes, interfaces of a component, its activators (if any) and its manifest are archived in a jar file. For example Figure 4.7 shows *component1*structured as a bundle (bundle 1). This bundle consists of Classes C and F, its unique provided and its interface (*Interface C, its adapt C*), and its facade classes (A and E). Suppose, this component has entry point (main () method of C), then the class *Component1Activator* was created and added to the bundle.

3.2.4 Guidelines for Implementing Components in OSGi Framework

The Open Services Gateway Initiative (OSGi) is a framework that supports the implementation of component-based, service-oriented applications in Java. Following are some steps for implementation.

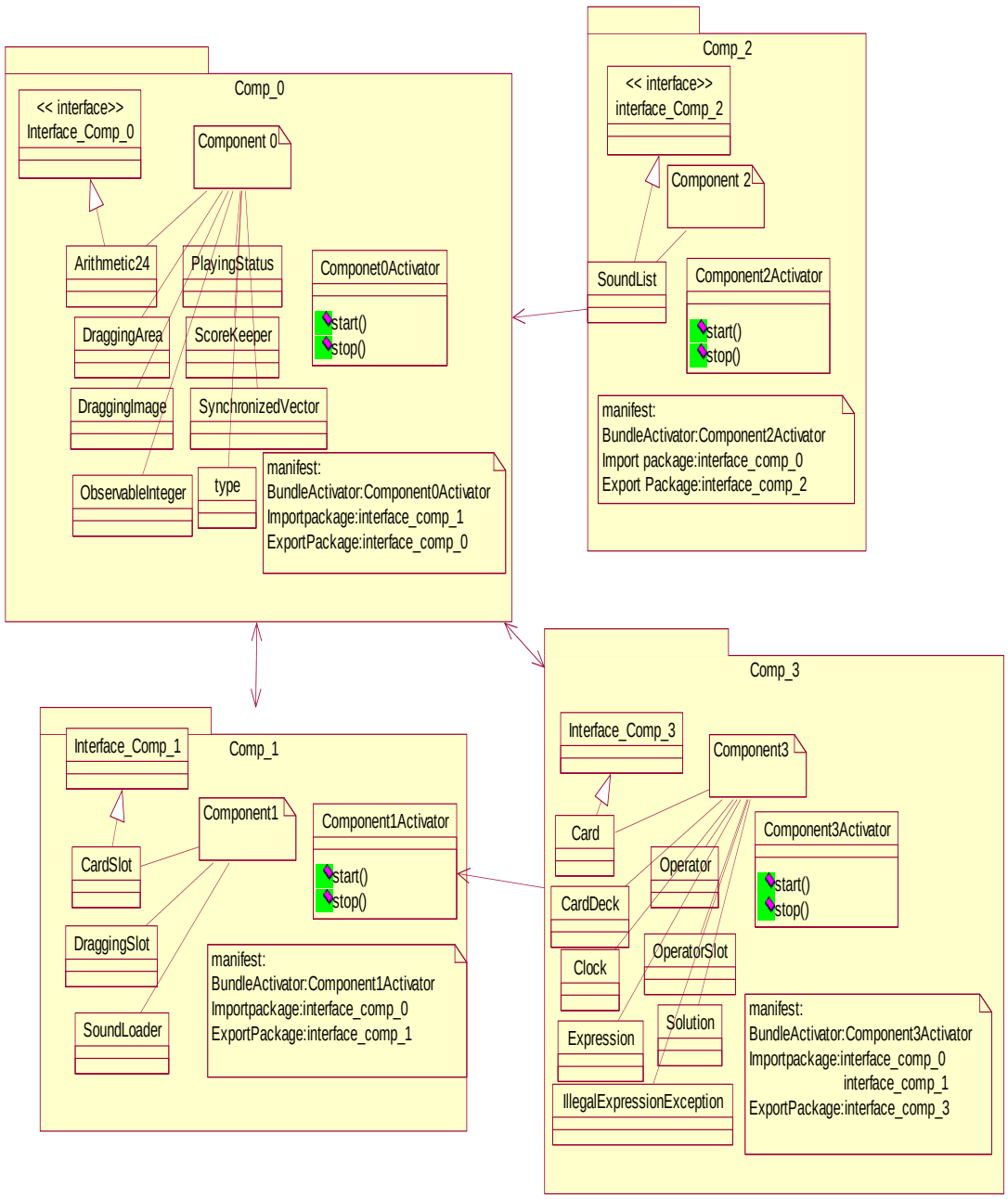
Steps:

- Let C be component based system consisting of components c1, c2,…..cn.
- For each component ci in C create separate bundle in the form of package consisting of classes of component ci.
- Add of interfaces of the component ci in the bundle. These interfaces plays role of required interfaces for component cj in C provided interface for ci.
- Add component activator class for component ci as a *BundleActivator*

- Finally the classes of component ci, interfaces of a component ci, its activators (if any) and its manifest are archived in a jar file.

Example: Suppose for ‘Arithmetic 24’ game application described in section 3.6.3 four components are created namely Component0, Component1, Component2 and component3. These components can be packed into bundles along with their interfaces and manifest for implementing it into OSGi framework. Following figure 3.6 shows representation of bundles for ‘Arithmetic24’ game application.

Figure 3.7: Representation of bundles for 'Arithmetic24' game application.



In this way created components through the proposed tool can be implemented in

Component based framework.

3.3 Selection of Clustering Process for the Methodology

The clustering techniques can be used effectively to facilitate software Architecture recovery. The clustering concepts required for the proposed methodology are presented in appendix II-(b). In this section we present selection of clustering process for the methodology.

With the objective of taking advantage of the features of the hierarchical clustering, in this study, Hierarchal clustering based approach is used to economically determining reusability of software components in existing object oriented systems.

3.3.1. Identification of Features and Entities in the System

The present study works on object oriented system, hence object or classes are the entities for our approach, as object are basic units for object oriented system.

3.3.2. Selection of Similarity Measure

The similarity measures and linkage method are the most important factors in agglomerative hierarchical clustering algorithm. The choice of a proper similarity measure and linkage method has even more influence on the clustering results. The quantitative computations of the similarities between classes can differ according to the measure. We will adapt the generic cohesion measure introduced by Frank [19] that is connected with theory of similarity and dissimilarity. A generic cohesion concept is applicable to different abstraction levels of software. It can be applicable to object oriented systems also. “Cohesion refers to the degree to which module components belong together”[19]. The distance measure supports the measurement of cohesion. Hence, cohesion measure is appropriate for our approach. The distance function d can be calculated for similarity measure. Most commonly used distances in object oriented

system are distance measured through method coupling i.e. usage relationship, distance measured through composition coupling and distance measured through inheritance coupling. We have proposed integrated coupling of these three couplings and used as distance measure, for agglomerative clustering algorithm.

3.3.3 Selection of Clustering Algorithm

The common process of clustering starts by parsing the source code of legacy system and then organizing the source code into cohesive sub systems that are loosely connected by particular algorithm. In the proposed approach, parsing source code of java applications for software architecture recovery takes place, as in terms of object oriented system terms, a component consists of a set of member classes and interfaces which specify their services.

The algorithm chosen here is agglomerative hierarchical clustering algorithm (AHCA) for component identification because it has following are advantages.

- A multi-level architectural view produced by agglomerative hierarchical clustering algorithms facilitates architectural understanding [60].
- They are non-supervised. They do not need extra information such as the number of expected clusters and candidate regions of search space for locating cluster.
- AHCA provides a view for software clustering; the earlier iterations presents the detailed view of the software architecture and the later ones presents a high-level view.
- For AHCA it requires entities to be clustered. We are using source code of java application as input; it is easy to treat classes as entities to be clustered.
- AHCA can produce a hierarchical decomposition for software system without defining the number of components in advance.

Selection of Linkage Method

Jain A.M. [40] suggested that during clustering the similarity between the newly formed and existing components should be iteratively recalculated. There are various linkage methods like single linkage, complete linkage, and average linkage. Most popular hierarchical clustering algorithms are variants of single linkage or complete linkage. These two algorithms differ in the way they characterize the similarity between a pair of clusters. In the single-link method, the distance between two clusters is the minimum of the distances between all pairs of patterns drawn from the two clusters (one pattern from the first cluster, the other from the second).In the complete-link algorithm, the distance between two clusters is the maximum of all pair wise distances between patterns in the two clusters. In either case, two clusters are merged to form a larger cluster based on minimum distance criteria. The complete-link algorithm produces tightly bound or compact clusters. The single-link algorithm, by contrast, suffers from a chaining effect. It has a tendency to produce clusters that are straggly or elongated. The clusters obtained by the complete link algorithm are more compact than those obtained by the single-link algorithm. The single-link algorithm is more versatile than the complete-link algorithm, otherwise. For example, the single-link algorithm can extract the concentric clusters but the complete-link algorithm cannot [40]. So we have decided to use single linkage in our approach.

3.3.4 Selection of Evaluation Criteria for Assessment of Components

In present study, we will focus on internal assessment and use evaluation metrics for components based on size of component and coupling between components. The approach will propose metric for cohesion within component. Using these metrics quality of components created through proposed method will be evaluated.

Chapter 4

Proposed Component Based Software Architecture Recovery

4.1 The Proposed Component Based Software Architecture Recovery Approach

Components are regarded as being more course-grained compared to traditional reusable artifacts such as objects and provide high level representation of the domain. Components can be used more effectively and are better suited for reuse than using objects. Creating reusable components from object oriented system is major task in migrating to component based system and it is one of the most prominent maintenance objectives to migrate systems to distributed computing environments using components. The objective of this research is to develop automatic approach which requires less human intervention to recover components and interfaces from object oriented system. To prove that existing legacy object oriented code can be reusable when industry migrates into new technology like component based. This obviously reduces the cost to company who wants to migrate to new technology.

The proposed work is aimed at a legacy object oriented system where the design documents are not available. We also aim to demonstrate how the proposed approach will help in identifying components and connectors from legacy object oriented system. This work proposes agglomerative clustering algorithm and uses size and coupling of component quality metrics to evaluate the quality of identified components and proposes component cohesion metric. The work also provides interface details using which interface packages or connectors can be prepared. This **work does not propose** creation

of reusable library for component and connector storage, as identified components and connectors can be stored in the library according to the component based framework the company uses. Instead, we provide guidelines for implementing components and interfaces in OSGi component based framework. The user should be familiar with component based Architecture. It is assumed that user knows java technology and wants to migrate from java classes to java components.

The user here is assumed as software maintainer or software engineers, who want to develop component based application using existing object oriented system. The goal is to achieve migration to component based software from existing object oriented system with minimum cost by reuse existing application. The user knows how to import existing java code into tool and use the results from the tool to create components and connectors. The user also must know dependency files of legacy source code and where to load that in the tool. For example if object oriented application contains servlet pages the servlet.jar file must be loaded while executing in the tool.

Present study shows the components and interface details retrieval by doing the following:

- Component Based Architecture Recovery from Object Oriented System from Existing Dependencies among Classes-

  The proposed process is based on the identification of source code entities and the relationship between them. The list of possible relationships between object oriented systems includes inheritance, composition, invocation relationship etc.

- Component Identification from Existing Object Oriented System using Hierarchical Clustering-

  A component is group of classes collaborating to provide a function of application. We need to group the classes based on similarity to generate component based system from existing object oriented system. Each of the group becomes component. A clustering algorithm allows grouping of classes of the application.

- Component Evaluation and Component Interface Identification from Object Oriented System-

  Identified group of classes working together will form components. We also need to identify required and provided interfaces to describe how they bind together.

4.2 Rationale for Component Based Software Architecture Recovery

Software Architecture modeling and representation is very important in software development process. Software Architecture provides high level view which is very useful in all phases of software life cycle like coding, maintenance, testing, etc. Component based software architecture is beneficial as it is useful for reusing system parts represented as components. Most of the existing object oriented systems do not have reliable software architecture and some legacy systems are designed without software architecture design phase. So by developing tool we can retrieve component based software architecture. The software architecture of the system is described as a collection of components along with the interaction among these components, where as the main system functional block are components, strongly dependent on connectors – which is abstraction capturing nature of these interactions. Therefore, the proposed work will focus on extracting component and interface details in component based architecture from existing object oriented system. As object-oriented development had not provided extensive reuse and computing infrastructures are evolving from mainframe to distributed environments, where objects technology has not led to massive development of distributed systems. However, component-based technology is considered to be more suited for distributed system development due to its granularity and reusability.

Using Component based software architecture is beneficial because-

- Exchange between software architects & programmers easily.
- Useful for reusing system parts represented as components.
- Clear separation between components & connectors.
- Localizing software defects & reducing risk of misplacing new functionalities during maintenance & evolution phases.

Software architecture has put forward connectors as first-class entities to express complex relationships between system components. Although components have always been considered fundamental building blocks of software systems, the way the components of the system interact may also be a determinant on the system properties. Component interactions were also recognized to be first class entities & architectural connectors have emerged as a powerful tool for supporting the description of these interactions. Components address only one aspect of large-scale development. Another important aspect is interaction among components.

Component contains only the business logic and communicates with one another only via well-defined interfaces the communication paths among the components are in modern component systems realized by software connectors, which allows explicit modeling of communication and also its implementations at runtime.

Major works have been proposed in the literature to recover component based Architecture, most of them are manual or semiautomatic, which requires other guidelines like design documents, human domain experts etc. Most of the approaches focus on component retrieval only and not about interface details i.e. connector classes.

To deal with this problem we have proposed approach of component based architecture recovery which aims to extract component based architecture from existing object oriented system using existing dependencies among classes and agglomerative hierarchical clustering algorithm. This approach is useful when no documentation of an application is available, and it requires very less human intervention.

4.3 The Proposed Framework and Tool

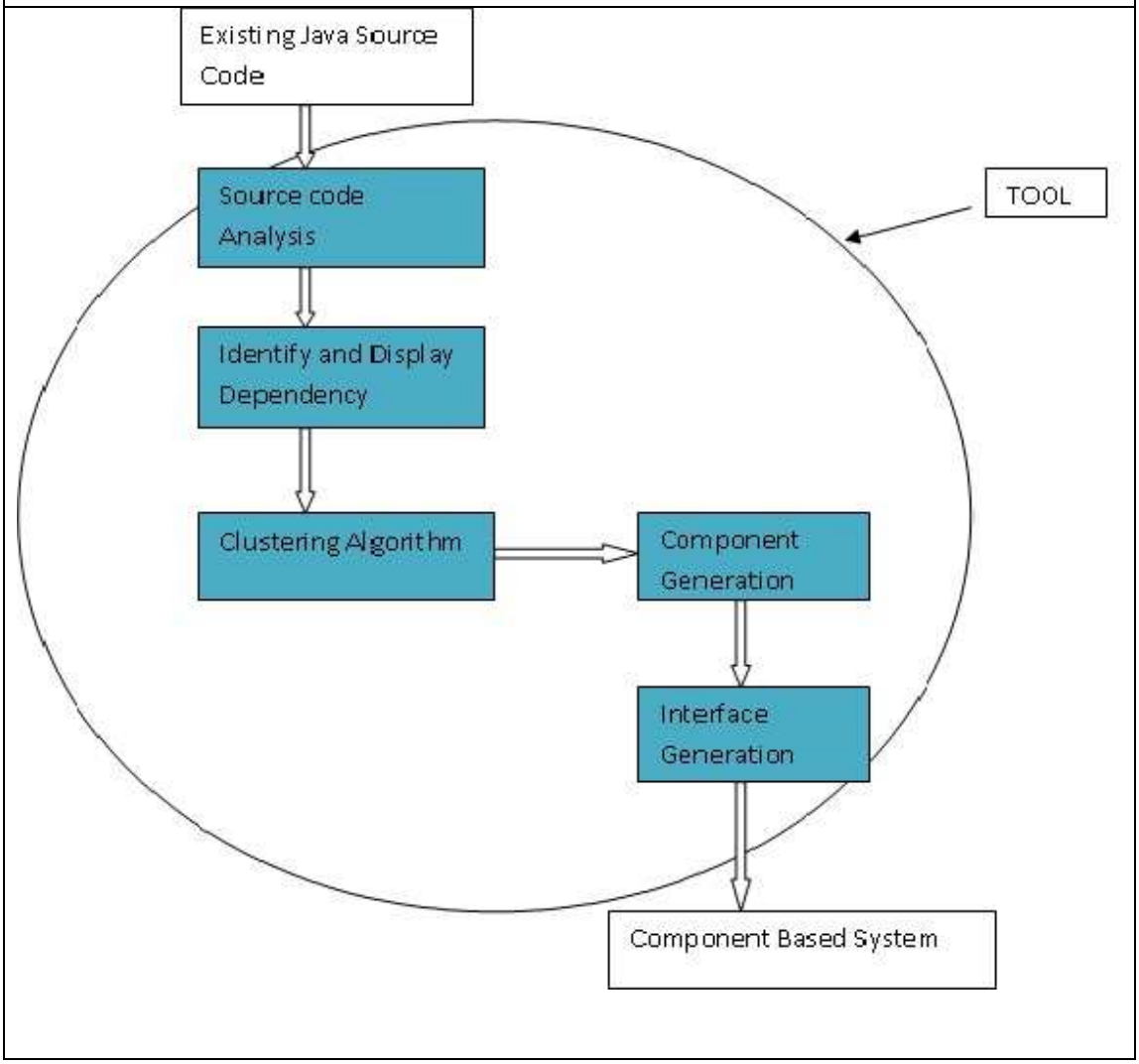
The framework is divided into three modules. We develop a tool consisting of these three modules for component identification and interface details generation. Following sections gives details of overall process and proposed tool.

We have identified three steps to produce a component based architectural view from an object-oriented application in proposed approach

- Identify Dependencies in Existing Object Oriented System
- Identify Components
- Component Evaluation and Interface Identification.

Figure 4.1 shows proposed approach for producing component based architecture.

Figure 4.1: Proposed Framework and Tool



4.3.1 Identify Dependencies in Existing Object Oriented System

We examine existing object oriented system to identify dependency among the classes using method coupling, inheritance coupling and composition coupling. We have evaluated the feasibility on Java software. Component-based software architecture is a high level abstraction of a system using the architectural elements: components which describe functional computing, connectors which describe interactions and configuration which represents the topology of connections between components.

Frank Simon et al [19] described, while recovering software Architecture from object oriented system different abstraction levels can be considered e.g. method level, variable level, object level and system level. Extensive literature research has justified these abstraction levels for software measures. Alae-Eddine et al [3] described the definition of metrics on Object Oriented system elements are obtained by identification of different types of relationship between different classes and computation of their strengths. Class coupling is one of the Object Oriented metric. Coupling is an indication of the connections between elements of the object oriented Design and indicates dependencies among classes. It is important to identify coupling for creating components. We need to determine precisely the dependency among classes and how to measure their strengths. The possible dependencies among Object Oriented system entities include inheritance, composition, aggregation and method invocations. So identifying these dependencies become the first step to recover software Architecture.

Jong kook Lee et al [45] described, well defined components designs are driven by a variety of factors e.g. the principles of cohesion and coupling are important factors for well-defined component design. Therefore in this study we are focusing on class coupling to identify well defined components.

Coupling is qualitative measure of the degree to which classes are connected to one another. Coupling is an indication of the connections between elements of the object oriented Design. It has been defined as a measure of the degree of interdependence between modules and the degree of interaction between modules. C. Rajaraman et al presented definition as "Coupling is a measure of the association, whether by inheritance or otherwise, between classes in a soft-ware product". Though coupling is a notion from structured design; it is still applicable to object-oriented design at the levels of modules, classes and objects. We are concerned only with coupling between classes. Thus, coupling indicates dependencies among classes. The possible dependencies between classes of object oriented system include inheritance, composition, method invocations

etc. Here, we are considering following important coupling dependencies as they are basis for identifying components from object oriented system.

Inheritance Coupling- Inheritance coupling is coupling between generalized class (Super class) and its specialized classes (Sub classes).

Composition Coupling-When instance of one class is referred in another class, then we have composition coupling.

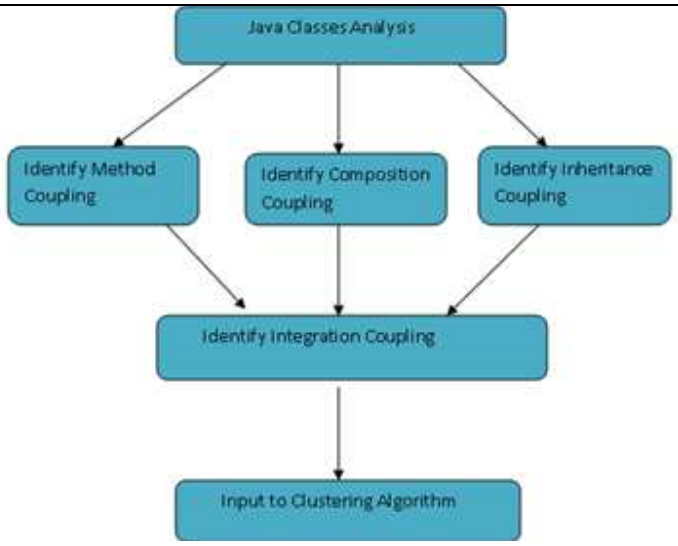
Method Coupling - When methods of one class use methods of another class hierarchy, then we have method coupling between the classes.

Integrated Coupling- It is a class's all three couplings inheritance coupling, composition coupling, and method coupling.

The proposed approach of extracting component based architecture from object oriented system is based on the identification of source code entities and the relation between them. The entities and relations have to be extracted by source code analysis and identify dependencies between the classes.

Figure -4.2 summarizes the process for identifying and displaying dependencies.

Figure 4.2: Process for Identifying Dependencies



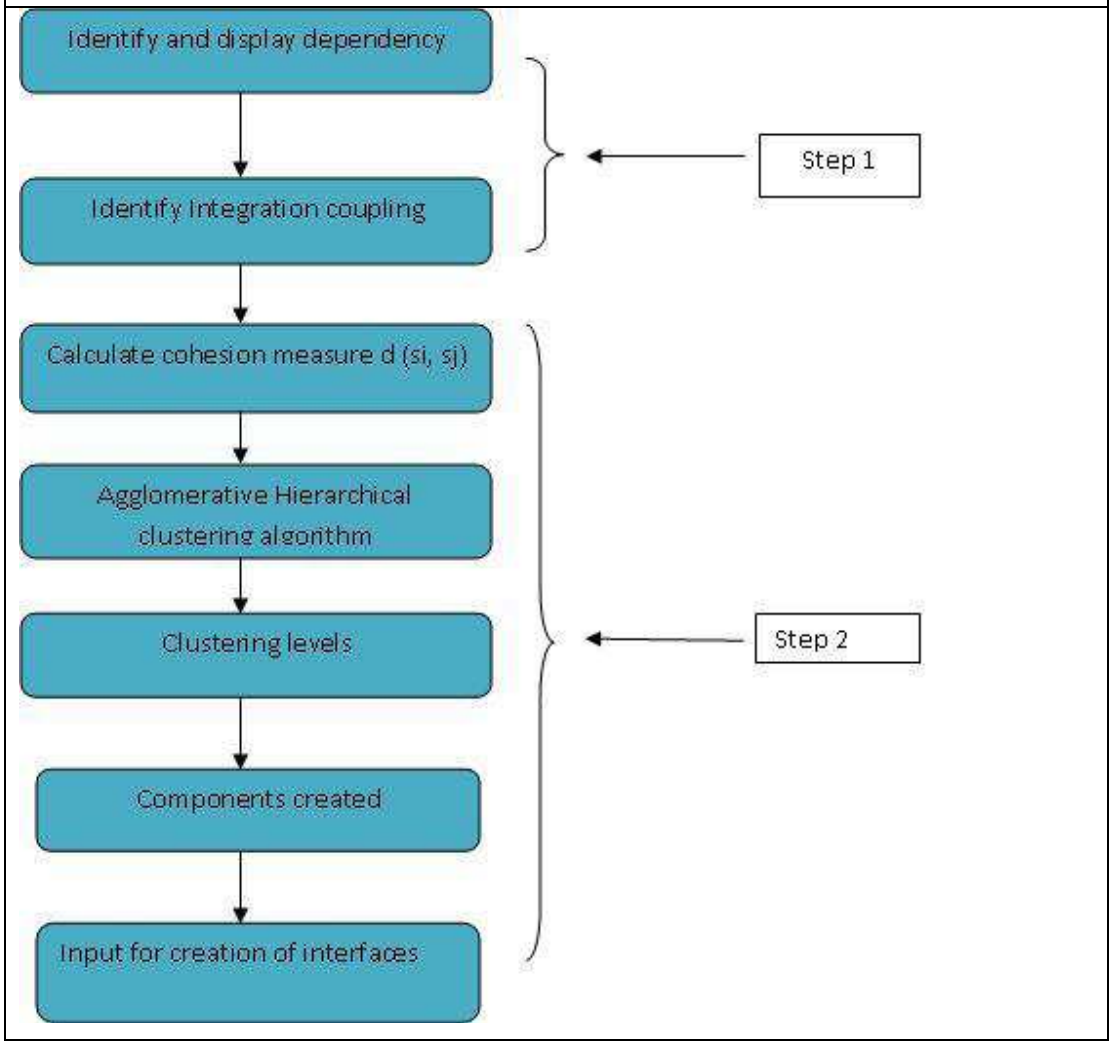
Integration coupling identified in this step is given as inputs to the next step i.e. identify components, which uses agglomerative hierarchical clustering algorithm to create components.

4.3.2 Identify Components

In this step we are generating input required for proposed algorithm; from integration coupling (step (i) above) i.e. similarity measure and distance function, d(Si, Sj) for proposed agglomerative clustering algorithm. The agglomerative hierarchical clustering algorithm identifies components from object oriented application. Cohesion measure distance function d(Si, Sj)gives distance between two classes Si and Sj of object oriented system S.

The process for identifying components is shown in figure 4.3 below.

Fig. 4.3: Process for Identifying Components



Following sub sections show how this distance function d (Si, Sj) is generated and the proposed algorithm.

Similarity Measure and Distance Function:

The most important factor in clustering process is similarity measure. Similarity measures determine how similar a pair of classes is. Similarity of classes can be calculated by variety of ways and choosing similarity measure is influence the result

than the algorithm. Theory of distance measure tells that the similarity between two things is the collection of their shared properties. If Si and Sj are two entities, then the distance measure holds the following statements.

1. d(Si, Sj) >= 0
2. d(Si, Si) = 0
3. d(Si, Sj) = d(Sj, Si)

We will adapt the generic cohesion measure introduced by Frank Simon [19] that is connected with theory of similarity and dissimilarity. Hence cohesion measure is appropriate for proposed approach. We consider distance d(Si, Sj) between two classes Si and Sj from S is expressed in the following expression (1) where S= {s1,s2,……..,Sn} be the set of objects to be clustered. Objective here is to group similar classes from S in order to obtain high cohesive groups (clusters).

$$d(si,sj) = 1-sim(si,sj) \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots(1)$$

Where,

$$sim(si,sj) = \frac{| b(si) \cap b(sj) |}{| b(si) \cup b(sj) |}$$

With b (Si):= {Pi ЄB| Si possess Pi},Pi – set of relevant properties of Si .So, distance measure focuses on the similarity measure of two entities with respect to a property subset B shown above. The distance function d(Si, Sj) is normalized between 0 and 1. The distance between two entities is larger the less similar they are, or vice versa. The two distinct entities can have a distance of 0.We have chosen distance between two classes as expressed in equation (1) because it emphasizes idea of cohesion. “Cohesion refers to the degree to which module components belong together” described Frank Simon [19]. So the equation (1) highlights concepts of cohesion. d is a semi-metric function so hierarchical clustering algorithm can be applied.

Distances in Object Oriented System:

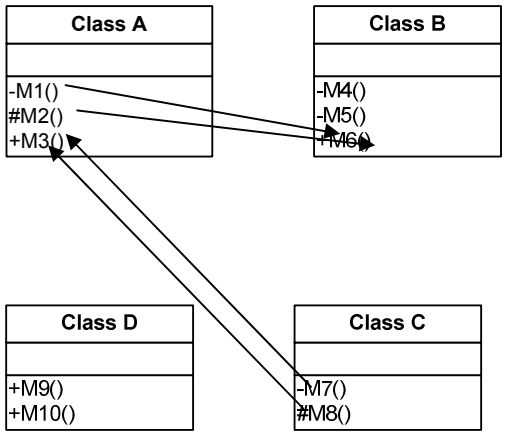
With respect to measurement theory and distance measurement distances can be calculated for pair of entities [19].In object oriented system several entity families can be detected based on abstraction levels. The abstraction levels can be variable level i.e. attributes of system, method level i.e. methods of system, object level i.e. classes of system and system level i.e. whole system. We are focusing on class level, because basic unit for object oriented system is class and we work on source code of object oriented system. We calculate here distance between two classes based on the relationship between classes Most commonly used distances in object oriented system are distance measured through method coupling i.e. usage relationship, distance measured through composition coupling and distance measured through inheritance coupling. We have proposed integrated coupling of these three couplings and used as distance measure, for Agglomerative clustering algorithm.

Distance through Method Coupling:

Method coupling - When methods of one class use methods of another class hierarchy, then we have method coupling between the classes.

Consider following example to illustrate calculation of the distance between source and destination class using method coupling.

Figure 4.4: Class Diagram with Method Coupling



In the above figure 4.4, private methodM1 () and protected method M2 () of class A, accesses public method M6 () of class B. Private method M7 () and protected method M8 ( ) of class C, accesses public method M3 ( ) of class A.

Thus, distance between two classes using method coupling can be calculated in following table.

Class	**Properties** (Methods in the class & accessed method by class)	**Dist (A, $)** = 1-(Intersection of properties of A &$)/(union of properties of A & $)	**Dist (B, $)** = 1-(Intersection of properties of B & $) / (union of properties of B & $)	**Dist (C, $)** = 1-(Intersection of properties of C &$)/(union of properties of C & $)	**Dist (D, $)** = 1-(Intersection of properties of D &$)/(union of properties of D & $)
A	M1, M2, M3, M6	1-4/4 = 0	1-1/6=0.833	1-1/6=0.833	1-0/6=1
B	M4, M5, M6	1-1/6=0.833	1-3/3=0	1-0/6=1	1-0/5=1
C	M3, M7, M8	1-1/6=0.833	1-0/6=1	1-3/3=0	1-0/5=1
D	M9, M10	1-0/6=1	1-0/5=1	1-0/5=1	1-2/2 = 0

Table 4.1 Distance Calculation using Method Coupling

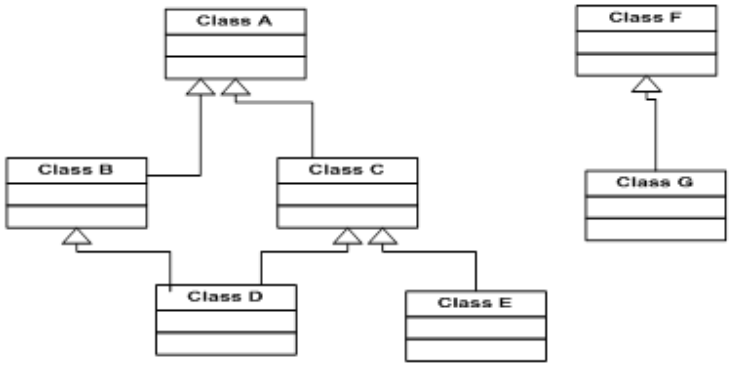
Thus, we have prepared distance matrix using method coupling of classes in object oriented system.

Distance through Inheritance Relationship:

Coad Yourdan [15] defined inheritance coupling which is coupling between generalized class (Super class) and its specialized classes (Sub classes) .The subclass has at least the same behavior as the super class. When looking at implementation level, the functionality of one class might be distributed over several classes from which it inherits. This is valid for all three kinds of inheritance. The entities of interest in this case are classes. Additionally for every class we are interested in its sub-classes and its super classes.

Consider following class diagram to illustrate calculation of the distance between source and destination class using inheritance coupling.

Figure 4.5: Class Diagram with Inheritance Coupling



In the given example this would be (extract):

{ (class A, class B, {class A →class A, class B →class B, class B →class A},

(class A, class D, {class A →class A, class D →class D, class D →class C, class D →class B,

class D →class A},

(class D, class E, {class D →class D, class D →class C, class D →class B, class D →A,

class E →class E, class E →class C, class E →class A},

(class A, class F, {class A →class A, class F →class F}

.... }

Thus, distance between two classes using inheritance coupling can be calculated in following table.

Class	**Properties** (Class & its parent class)	**Dist (A, $)**	Dist (B,$)	**Dist (C,$)**	**Dist (D,$)**	**Dist (E, $)**	**Dist (F, $)**	**Dist (G,$)**
A	A	1-1/1=0	1-1/2=0.5	1-1/2=0.5	1-1/4=0.75	1-1/3=0.66	1-0/2=1	1-0/3=1
B	B, A	1-1/2=0.5	1-2/2=0	1-1/3=0.66	1-2/4=0.5	1-1/4=0.75	1-0/3=1	1-0/4=1
C	C, A	1-1/2=0.5	1-1/3=0.66	1-2/2=0	1-2/4=0.5	1-2/3=0.33	1-0/3=1	1-0/4=1
D	D, B, C, A	1-1/4=0.75	1-2/4=0.5	1-2/4=0.5	1-4/4=0	1-2/5=0.6	1-0/5=1	1-0/6=1
E	E, C, A	1-1/3=0.66	1-1/4=0.75	1-2/3=0.33	1-2/5=0.6	1-3/3=0	1-0/4=1	1-0/5=1
F	F	1-0/2=1	1-0/3=1	1-0/3=1	1-0/5=1	1-0/4=1	1-1/1=0	1-1/2=0.5
G	G, F	1-0/3=1	1-0/4=1	1-0/4=1	1-0/6=1	1-0/5=1	1-1/2=0.5	1-2/2=0

Table 4.2 Distance Calculation using Inheritance Coupling.

Where,

Dist (A, $) = 1-(Intersection of properties of A & $)/(union of properties of A & $)

Dist (B, $) = 1-(Intersection of properties of B & $)/(union of properties of B & $)

Dist (C, $) = 1-(Intersection of properties of C & $)/(union of properties of C & $)

Dist (D, $) = 1-(Intersection of properties of D & $)/(union of properties of D & $)

Dist (E, $) = 1-(Intersection of properties of E & $)/(union of properties of E & $)

Dist (F, $) = 1-(Intersection of properties of F & $)/(union of properties of F & $)

Dist (G, $) = 1-(Intersection of properties of G & $)/(union of properties of G & $)

Thus, we have prepared distance matrix using inheritance coupling of classes in object oriented system.

In the same way we can prepare distance matrix for **composition coupling**, where properties would be the class (part class) and other class (whole class) whose attribute is used by the class. When instance of one class is referred in another class, then we have composition coupling, i.e. properties considered here for distance calculation are part class and whole class.

Similarly for **integration coupling**, we consider all the coupling with the class and prepare distance matrix using above technique. Thus equation (1) defined above evaluates all pair-wise distances between clusters, which is the most important cohesion metric (d (Si, Sj)), that will be used in proposed agglomerative hierarchical clustering algorithm. Thus, Construct distance matrix using distance value (using entity e Є S, p(e)- a set of relevant properties of e)

Propose Agglomerative Hierarchical Clustering Algorithm (AHCA):

Software system is composed of set of classes and dependencies among the classes. The semi metric function d(Si, Sj) is calculated using existing dependencies in object

oriented system, which is one of the important input to the clustering process. The clustering algorithm is written in chapter 5.

Finally using cluster levels components are created which can be used for creating interfaces of the components. Final cluster level-1 is used for creating components.

In this step Cluster levels created are used in creating components, are used as input to next step i.e. identify interfaces.

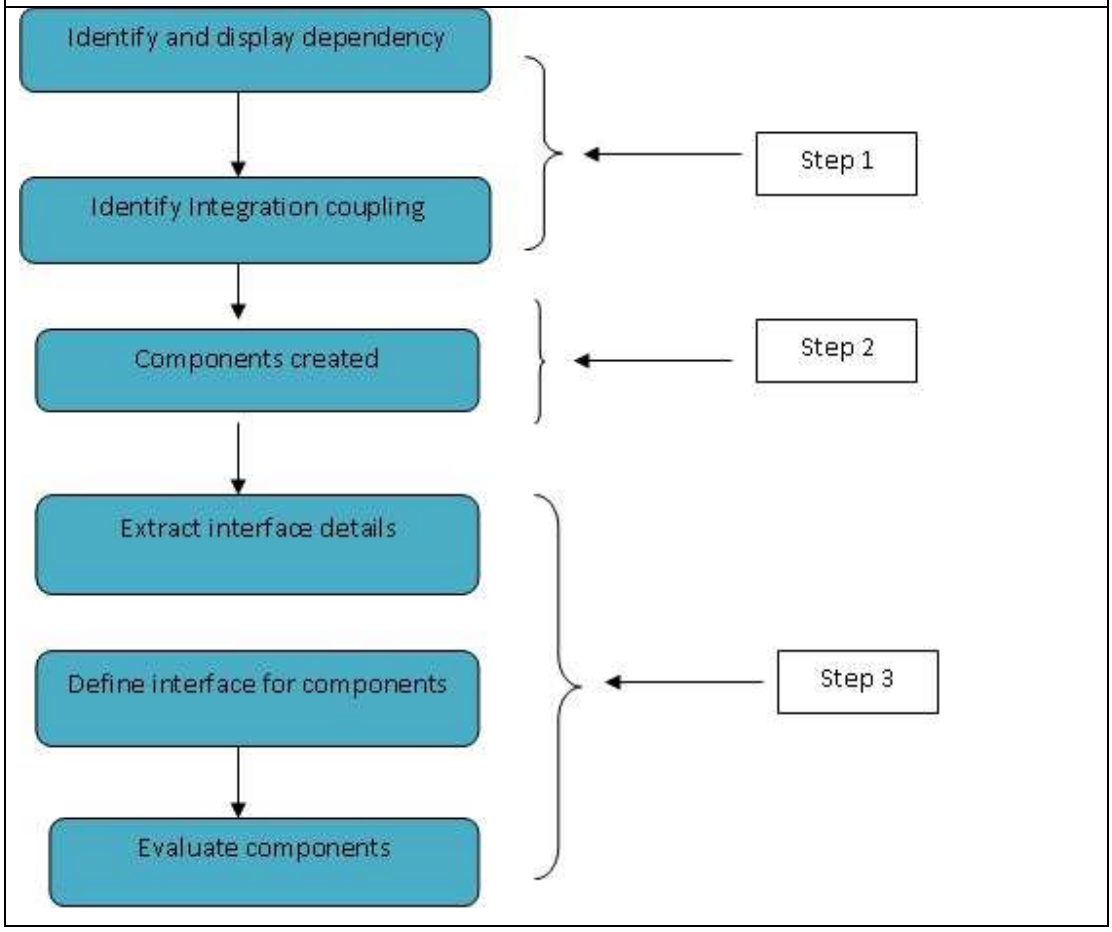
4.3.3 Component Evaluation and Interface Identification.

In this step, we will demonstrate how to extract interfaces among components and component evaluation while recovering component based architecture. Using the components generated in previous step, interface details are identified. Identified components are evaluated using component cohesion, component coupling, and component size metrics for quality of components. This step is further divided into two sub steps as Identify interfaces and component evaluation.

- **Identify Interfaces:**

Component based system consists of components and interfaces. Component interfaces are the means by which components connect with each other. A component interface specifies the service that the component provides and requires. Among all of the methods in the component, only public methods used from outside provide services to other components or classes. Therefore we create a provide interface that includes the public methods that exists in any of the component's classes and which are used by the outside of that component. Require-interface is the union set of every method in other components that is called by the component. To reduce cyclic dependency among components, we group these interfaces as packages. The process of identifying interfaces and component evaluation is shown in below figure. 4.6.

Fig. 4.6: Process for Identification of Interfaces and Component Evaluation



- **Component Evaluation:**

The component evaluation step above accepts the results produced through clustering i.e. components created, interfaces details created as input and evaluates the quality of identified components. There is several evaluation criterions proposed to qualify clustering results. The basic quality metrics to evaluate software system are coupling and cohesion, which can cause serious impact on maintenance, evolution, and reuse. Criteria for components used by us are size, coupling and cohesion. There should also be appropriate number of implementation classes in well-organized components.

Size:

Jian Feng Cui [42] proposed size as evaluation criteria to show well organized components with appropriate number of implementation classes. So using size we evaluate clustering results. According to them sum of ratios of single class component, classes in largest component and other intermediate components should be 100%.

- Ratio of Single class component=Number of Single class component/Total number of classes
- Ratio of classes in largest component=Number of classes in the largest component/Total number of classes
- Ratio of other intermediate components = Number of classes in intermediate components /Total number of classes

Coupling:

In component based system coupling shows how tightly one component is interacting with other components in the system. Coupled Component Ratio (CCR) is one of the metric for evaluating component coupling proposed by Jian Feng Cui [42]. According to them two components are said to be coupled if there is connection between them and CCR is defined as Number of components coupled with particular component/(Total no. of components in system – 1). The CCR value of component lies between 0 and 1.Smaller the CCR value better the component is.CCR value 1 indicates that component is coupled with all other components in system.CCR value 0 indicates that component is entirely independent.

Cohesion:

Cohesion in component based system is how tightly classes are coupled within the component. Cohesion metric is used to measure quality of components for reusability

and maintainability. We propose Component Cohesion Metric (CCM) as Number of component's self-couplings/Total number of couplings of that component. Where total number of couplings of component = self-coupling + coupling with other components within system. The value of CCM lies between 0 and 1. A higher CCM value indicates more similar behavior is grouped together i.e. more tightly coupled classes are grouped together.CCM value 1 indicates high cohesion within component.

Thus the tool identifies different kinds of dependencies among the classes then uses clustering algorithm to identify components. Interface details of the extracted components are identified by tool using which interface packages can be defined and components are evaluated based on component quality metrics size, component coupling and component cohesion.
We will evaluate the tool on java application as a case study to verify the results.

4.4 Summary

This chapter elaborates on the proposed framework of component based software architecture recovery. It describes in detail three steps of entire tool development process consisting of three modules. Module 1 for identifying dependencies in object oriented system. Module 2 for identifying components and finally Module 3 for component assessment for quality and interface identification. The chapter talks about calculation of similarity measure for clusters and distance calculation. It describes various types of distances in object oriented system like distance through usage relationship, distance through inheritance relationship, distance through composition relationship. These distances are used to calculate similarity measure for proposed Agglomerative Hierarchical Clustering Algorithm. The algorithm is used for component creation. It also describes about various quality metrics that will be used for component evaluation. Detail implementation of the proposed tool is presented in chapter5.

Chapter 5

Implementation of Proposed Component Based Software Architecture Recovery Framework

Introduction

Architecture recovery is a part of reverse engineering concerned with identifying architectural components such as subsystems, modules, objects as well as their interrelationships called connectors. Component based architecture recovery consists of identifying components and connector i.e. interfaces among the components.

Architecture recovery consists of detection of components and detection of connectors. Thus, the elements of Software Architecture are components and connectors.

Since industry is migrating from object oriented system to component based system as components more reusable and beneficial than objects, it is important to recover component based software architecture from object oriented system. It will help software developer or software maintenance person to create components and interface among these components, as a part of component based system.

In this chapter we will implement the proposed framework with an objective to help software developers to migrate the software in new working environments.

5.1 Implementation of the Proposed Framework and the Tool.

We are implementing proposed framework by developing a tool. We have developed this tool using Java platform under eclipse Galileo version with windows as operating system. The tool takes input as java source files with .java extension. The tool is developed in three modules.

5.1.1 Module 1:Identify Dependencies in Existing Object Oriented System

The first module identifies dependencies in existing object oriented system and displays result as Method coupling table, composition coupling table , inheritance coupling table, and integrated coupling table. These tables are nothing but the existing dependencies among the classes of object oriented source code given as input to tool. Let us assume that S is a object oriented system, consisting of n different classes s1,s2,s3,….sn. In this case S is legacy source code of java application, whose component based architecture needs to be identified.

Algorithm: 5.2.1 lists the pseudo algorithm for identifying dependency in existing object oriented system.

Algorithm: 5.2.1 Identifying dependency

Input: The object oriented software system S= {s1, s2, s3…sn} where s1, s2,…sn are classes of object oriented System and n is number of classes

Output: Tables showing source class and its coupling classes list

Method:

1. For each class s1 to Sn from S Do
2. Find method coupling with remaining S-s1 classes. If found return true, otherwise
   false.
3. Find composition relationship i.e. whole-part relationship, with remaining S-s1 classes. If found return true, otherwise false.
4. Find inheritance relationship i.e. parent-child, with remaining S-s1 classes. If Found return true, otherwise false.
5. Find integration coupling. If found return true, otherwise false.
   End for
6. Save these different couplings.
7. Display it in tabular format.

5.1.2 Module 2: Identify components

The second module is developed to identify components using these coupling tables. In this module, we are implementing agglomerative hierarchical clustering algorithm. The module calculates the distance function using coupling tables generated in module one. The distance calculation is required for agglomerative hierarchical clustering algorithm.

Using the integrated coupling table generated in module 1, we calculated semi- metric function d(si,sj) for software system S. The function d is normalized between 0 and 1. So the threshold chosen is 0.7 for similarity. Using these inputs to proposed clustering algorithm cluster levels are generated. Cluster level before final cluster level is used to create components.

The distance function equation (1) below, calculated in algorithm using integrated coupling is shown in Algorithm 5.2.2.

$$d(si,sj) = 1-sim(si,sj) \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots(1)$$

Where,

$$sim(si,sj) = \frac{|\, b(si) \cap b(sj) \,|}{|\, b(si) \cup b(sj) \,|}$$

sim (si, sj) = (Intersection of properties of si & sj)/(union of properties of si & sj)
Here, si and sj are classes from object oriented system S.
Properties in this case are:
- Methods in the class si& accessed method of sj by class si
- Class si & its parent class
- Class si (part class) and other class sj (whole class)
- Class si's method & methods of other classes sj accessed by si + Class si & its parent class + Class si (part class) and other class sj (whole class)

Algorithm: 5.2.2 Distance calculation

Input: Integration coupling from object oriented system S

Output: distance matrix (distance d(si,sj) = 1-sim(si,sj)), which displays source class, destination class, intersection count, union count of relevant properties and distance.

Method:

1. For each source class si and destination class sj in S do
2. Find union count using set of relevant properties of si
3. Find intersection count set of relevant properties of si

   If union count =0 or intersection count =0 then

   distance d(si,sj) =1.

   Else

   distance d(si,sj)= 1-(intersection count)/(union count)

   end for
4. Display distance matrix in tabular format.

Algorithm: 5.2.3 lists the pseudo algorithm for agglomerative hierarchical clustering algorithm, which takes input classes from object oriented system S consisting of n number of classes, semi metric function d(si,sj) calculated in Algorithm 5.2.2 and threshold value for clustering which is chosen as 0.7, as distance function is normalized between 0 and 1.Let us assume that P is number of clusters. Initially number of clusters is equal to number of classes n in the object oriented system S. Each cluster contains corresponding class. i.e. each cluster contains single class initially. Thus c1,c2,c3…cp be clusters in the system C. We evaluate all pair wise distances between clusters and construct distance matrix using relevant properties P of each class si in system S mentioned above. Then look for the pair of clusters with shortest distance and remove the pair from the matrix and merge them. We evaluate all distances from this new cluster to all other clusters and update the distance matrix. We continue with this process until we get the distance matrix reduced to a single element.

Algorithm: 5.2.3 Agglomerative hierarchical clustering algorithm (AHCA)

Inputs: - The object oriented software system S= {s1, s2, s3…sn} where s1, s2,…sn are classes of object oriented System and n is number of classes, the threshold chosen is 0.7 and the semi- metric function d between entities.
Output:-clusters at different level
Algorithm:

P=n; //initial number of clusters
For i=1 to n
Ci = {si}
End for
C = {c1, c2…cp} // clusters in the system
Repeat

- d(ci,cj)=1-sim(ci,cj) // Evaluate all pair wise distances between clusters
- Construct distance matrix using distance value (using entity e Є S, p(e)- a set of relevant properties of e)
- Look for the pair of clusters with shortest distance.
- Remove the pair from the matrix and merge them.
- Evaluate all distances from this new cluster to all other clusters and update the matrix.

Until the distance matrix is reduced to a single element.

Thus, cluster levels are created using agglomerative clustering algorithm and using one level before final level of clusters components are created.

5.1.3 Module 3: Component Evaluation and Interface Identification.

The third module is developed for Component evaluation and interface identification. Components created by module two are evaluated for quality. We are implementing proposed quality metrics in the tool. The tool displays coupling among components and cohesion within the component. Here C is the component based system consisting of n components. Let us say C={c1,c2,c3,…….,cn}. After Algorithm 5.2.3 we get these components, c1,c2,…etc. The quality metric size says that sum of all the ratios should be 1 to show that all the components are well created with appropriate number of classes.

All these component ratios contain ratio of Single class component, ratio of classes in largest component and ratio of other intermediate components.

Algorithm: 5.2.4 lists the pseudo algorithm for evaluating size metric of component.

Algorithm:5.2.4 Component evaluation using size metric

Input: component based system C={c1,c2….cn},migrated from object oriented system S

Output: Ratio of Single class component, Ratio of classes in largest component, Ratio of other intermediate components

Method:

1. For each component ci from C
   Find Total number of classes in ci
   end for
2. Find Number of Single class component from C
3. Find Number of classes in the largest component
4. Find Number of classes in intermediate components
5. Calculate Ratio of Single class component=Number of Single class component/Total number of classes
6. Calculate Ratio of classes in largest component=Number of classes in the largest component/Total number of classes
7. Calculate Ratio of other intermediate components = Number of classes in intermediate components /Total number of classes
8. Calculate Ratio of all components = (Ratio of Single class component + Ratio of classes in largest component + Ratio of other intermediate components)
9. If Ratio of all component = 1 then
   components are well organized with appropriate number of implementation classes.
   Else
   Components are not well organized.
10. Display ratio of all components.

The second metric we used here is component coupling metric. We evaluate component

coupling values for each component in the component based system C. We calculate Coupled Component Ratio (CCR), which shows how tightly component ci from C is coupled with other components cj in the system C. Smaller the CCR value lower coupling and higher the CCR value tight coupling.**Algorithm:5.2.5** lists the pseudo algorithm for evaluating component coupling metric.

Algorithm:5.2.5 Component evaluation using Coupling metric

Input: component based system C={c1,c2….cn} ,migrated from object oriented system S

Output: coupling value of all components

Method:

1. For each component ci from C check if ci is connected to cj
2. If true, then

   Find Number of components coupled with component ci

   Find Total no. of components in system C.
3. Calculate Coupled Component Ratio (CCR) // CCR value lies between 0 and 1

   CCR = Number of components coupled with component ci /(Total no. of components in C system – 1)
4. If CCR value is smaller then lower coupling

   Else

   Higher coupling between components.
5. Using CCR display coupling among the components in system C.

Before implementing our third evaluation metric cohesion, we need to identify interface details among the components, because these details are used for evaluating cohesion metric of component. **Algorithm: 5.2.6** lists the pseudo algorithm for identifying interface details among the components created using **Algorithm 5.2.2 and Algorithm 5.2.3.** Here we identify method coupling of each component ci with remaining C-ci components and coupling with ci itself, composition coupling of component ci , with

remaining C-ci components and coupling with ci itself also inheritance coupling of component ci with remaining C-ci components and with itself. Coupling with itself means how classes in each component interact with each other. Coupling with other classes means how classes from component ci interact with classes from component cj. These couplings are nothing but interfaces among all the components from c1 to cn, which are used to evaluate cohesion within each component ci of C, as well as it will work as required and provided interfaces of each components in system C. These required and provided interfaces are nothing but the connectors of components.

Algorithm:5.2.6 interface details identification

Input: component based system C={c1, c2….cn} ,migrated from object oriented system S

Output: Displays components along with their interaction coupling and coupling type

Method:

1. For each component ci in C
2. Find method coupling of component ci with remaining C-ci components and with ci itself.
3. If found return true, otherwise false. end if
4. Find composition coupling of component ci , with remaining C-ci components and With ci itself.
5. If found return true, otherwise false. end if
6. Find inheritance coupling of component ci with remaining C-ci components and with itself.
7. If found return true, otherwise false. endif
8. End for
9. Save these interface details of all components within system C
10. Display interface details in tabular format.

The third evaluation metric we have proposed in chapter 4 is Component cohesion metric (CCM) for evaluation of component. We take coupling details of component from **Algorithm 5.2.6 and** components from C created in **Algorithm 5.2.2 and Algorithm**

5.2.3 as input to **Algorithm 5.2.7.**The value of CCM lies between 0 and 1.If CCM value is larger higher is cohesion. i.e. more tightly coupled classes are grouped together and if CCM value is lesser, lower cohesion within the component. The component is said to be better component if it has maximum cohesion and less coupling with other components in the system.

Algorithm:5.2.7 Component evaluation using cohesion metric

Input: component based system C={c1,c2….cn} ,migrated from object oriented system S. Interface detail table generated, which shows component ci connected with cj by which coupling type in system C.

Output: Cohesion within component.

Method:

1. Find each component ci's self-coupling count in system C
2. Find count of component ci's coupling with cj, …cn.
3. Calculate total number of couplings of component ci = self-coupling of ci + coupling with other components within system C.
4. Calculate Component Cohesion Metric (CCM) // CCM value lies between 0 and 1
   CCM = Number of component ci's self-couplings/Total number of couplings of that component ci.
5. If CCM is higher then
   Higher cohesion within component
   Else
   Lower cohesion within component
6. Using CCM display cohesion within component

5.2 Summary

The main objective of conducting the research study at this juncture was to present various algorithms such as identifying dependency among classes, distance calculation for similarity among classes, agglomerative hierarchical clustering algorithm, component evaluation using size metric algorithm, using coupling metric algorithm, interface details identification algorithm and component evaluation using cohesion metric algorithm. These algorithms are implemented successfully and results obtained are presented in chapter 6. In this way proposed framework of three modules are implemented successfully.

Chapter 6

Results and Analysis

The research study “Extraction of connector classes from object oriented system while recovering software architecture” comprises of three modules. Before starting the first module, we need to identify static structure of object oriented system. This can be done by retrieving class diagram of the object oriented system. For retrieving class diagram, we have examined four existing reverse engineering tools - IBM Rational Rose, Enterprise Architecture, Reverse and ArgoUML. Post this we have compared results from them and selected a tool which retrieves maximum static information. Chapter 3 elaborates more details on these reverse engineering tools. Since we are recovering software architecture of a system whose design documentation is not available, this static information retrieved from reverse engineering tool is used to compare the results from module-1 of proposed approach.

Module-1 constitutes identifying existing dependencies in the object oriented system. Existing relationships in the object oriented code helps to group the related classes together in the form of components hence this module is designed and implemented. Here we have considered important relationships in any object oriented systems i.e. inheritance relationship, composition relationship and method calls from one class to other classes in the system.

Module-2 constitutes of identifying components from object oriented system. We have implemented similarity distance calculation algorithm and agglomerative clustering algorithm to group similar classes into one component. We have used 6 small and

medium size object oriented applications to test , how proposed tool creates components based on the existing relationships in the object oriented application.

Module-3 constitutes identifying interface details of component identified on module-2.These interface details are used to create connectors of components i.e. required and provided interface of components. These required and provided interfaces help to components to communicate with each other. The result and analysis chapter talks about all the three modules of proposed and implemented approach. It also talks about the static information retrieved from existing reverse engineering tool. This research work has been published in various International Journals and Conferences. The proposed approach focuses more on the small and medium size applications. The next section elaborates on the results derived during the various phases of the research.

Case study: We have chosen small java software. 'Arithmetic24 Game'. This is a software game application developed in Java by Huahai Yang. It is a simulation of popular traditional card game. It consists of 19 classes and 1 interface.
Following sections presents the results using the same case study.

6.1 Module 1: Identify Dependencies in Existing Object Oriented System

Objective:

To find inheritance coupling, composition coupling and method coupling and integrated coupling of these three couplings. We consider these important coupling dependencies as they are basis for identifying components from object oriented system. Components are required to create meaningful connectors.

Strategy :

The proposed approach of extracting component based architecture from object oriented system is based on the identification of source code entities and the relation between them. The classes and relations have to be extracted by source code analysis and identify

dependencies between the classes. Input all .java files of Arithmetic 24' game to the proper directory in the tool and execute main program of the tool.

Algorithms implemented: Algorithm: 5.2.1 Identifying dependency described in chapter 5 was implemented. Also some supporting programs are written and executed to identify different dependencies among classes.

Results from Module 1

When 'Arithmetic 24' game's source code is given as input to the first module, output shows that all the classes are extracted by module1 of proposed approach along with different coupling tables. **Results from module 1** are shown in figure 6.1 and figure 6.2.

Figure 6.1: Method, Composition, Inheritance Dependency identified from Proposed Approach & Tool of 'Arithmetic24' Game software

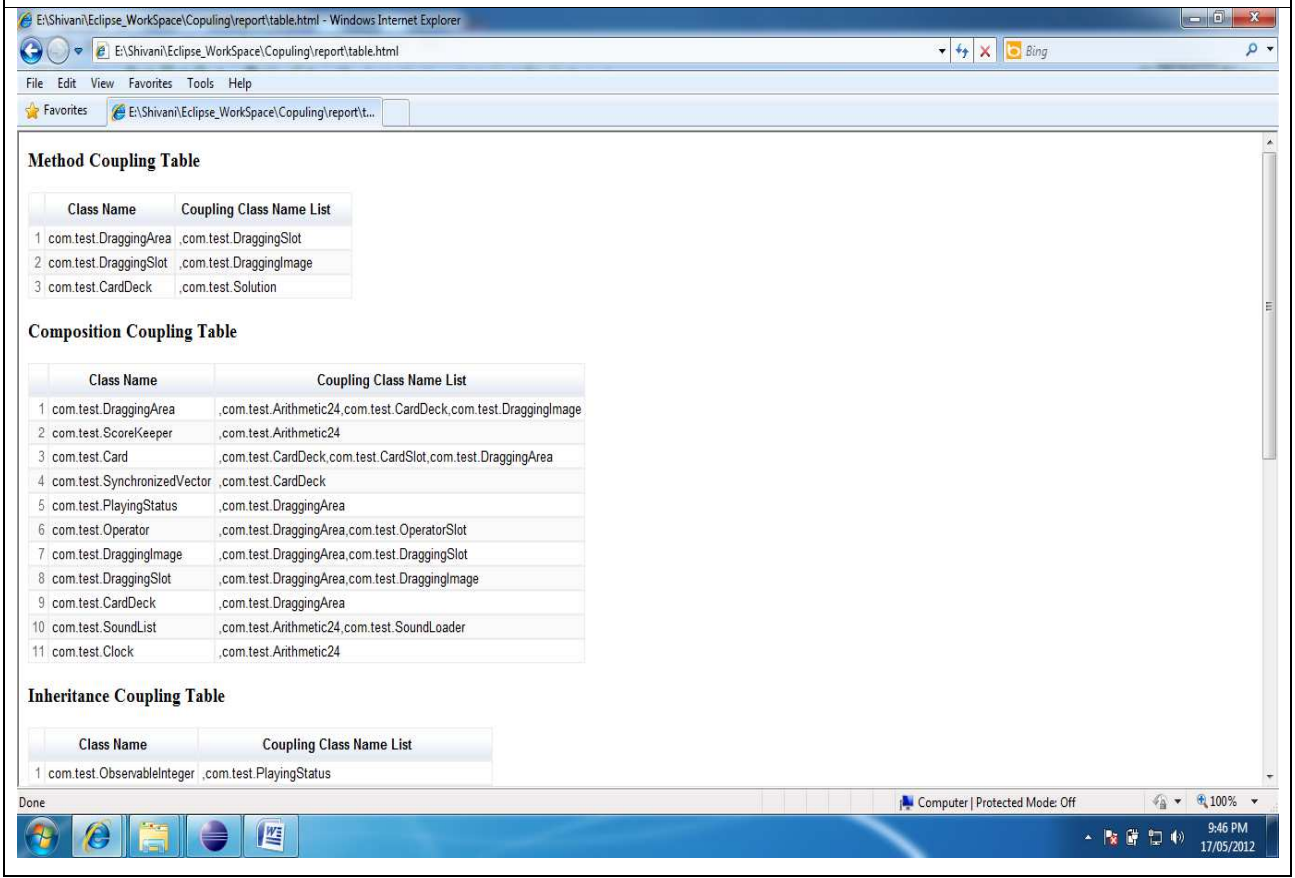
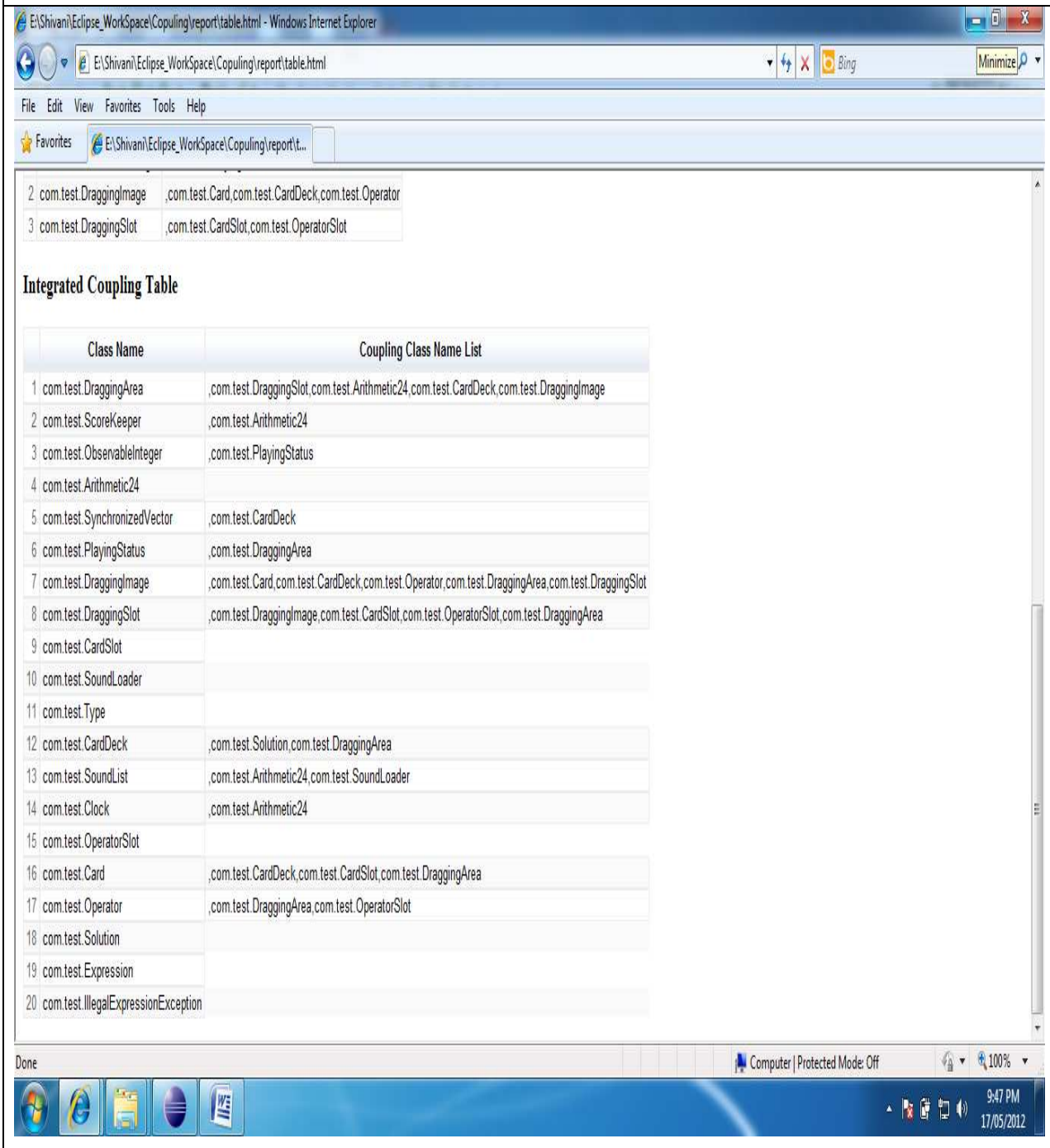


Figure 6.2: Integrated Coupling identified from Proposed Approach & Tool of Arithmetic24 Game software



6.2 Module 2: Identify Components

Objective:

To group the similar classes together to form the components using existing dependencies among classes.

Strategy :

Using the existing dependencies among the classes i.e. output generated in Module-1 shown in figure 6.1 - 6.2 and apply similarity distance algorithm, **Algorithm 5.2.2** and Agglomerative clustering algorithm**, Algorithm 5.2.3**and create cluster levels. Using final cluster level – 1,create components.

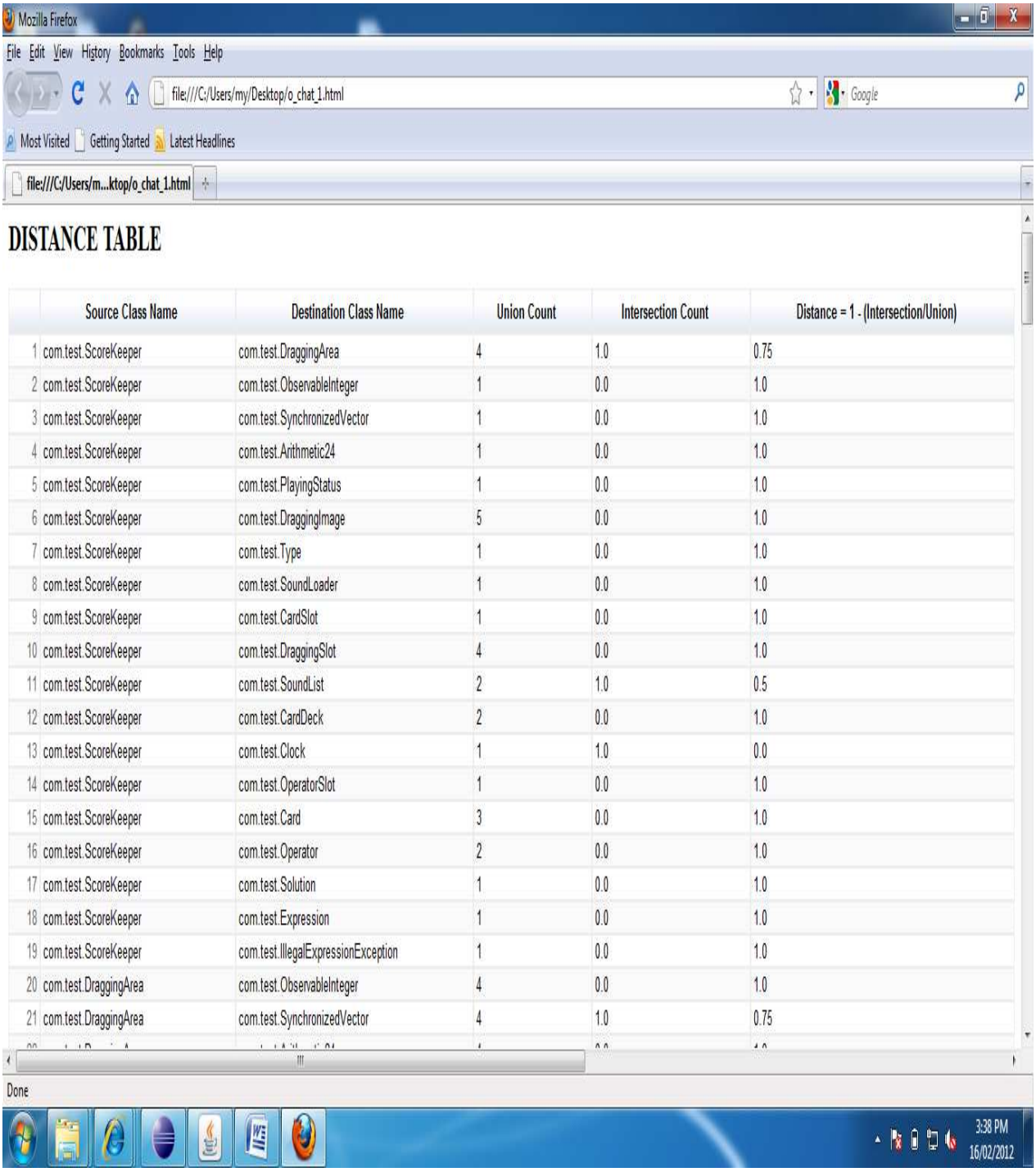
Algorithms implemented: Algorithm: 5.2.2 Distance calculation and Algorithm: 5.2.3 Agglomerative hierarchical clustering algorithm (AHCA)described in chapter 5 were implemented. We have also written and executed some supporting programs to identify components.

Results from module 2:

Using the integrated coupling table shown in figure 6.6we calculated semi- metric function d(si,sj) for software system S. The function d is normalized between 0 and 1. So the threshold chosen is 0.7 for similarity. Using these inputs to proposed clustering algorithm, we got cluster levels from 0 to 4 which are figures 6.8, figure 6.9 and figure 6.10 shows components created from the module 2.Distance tables created using integrated coupling is shown in figure 6.7 below.

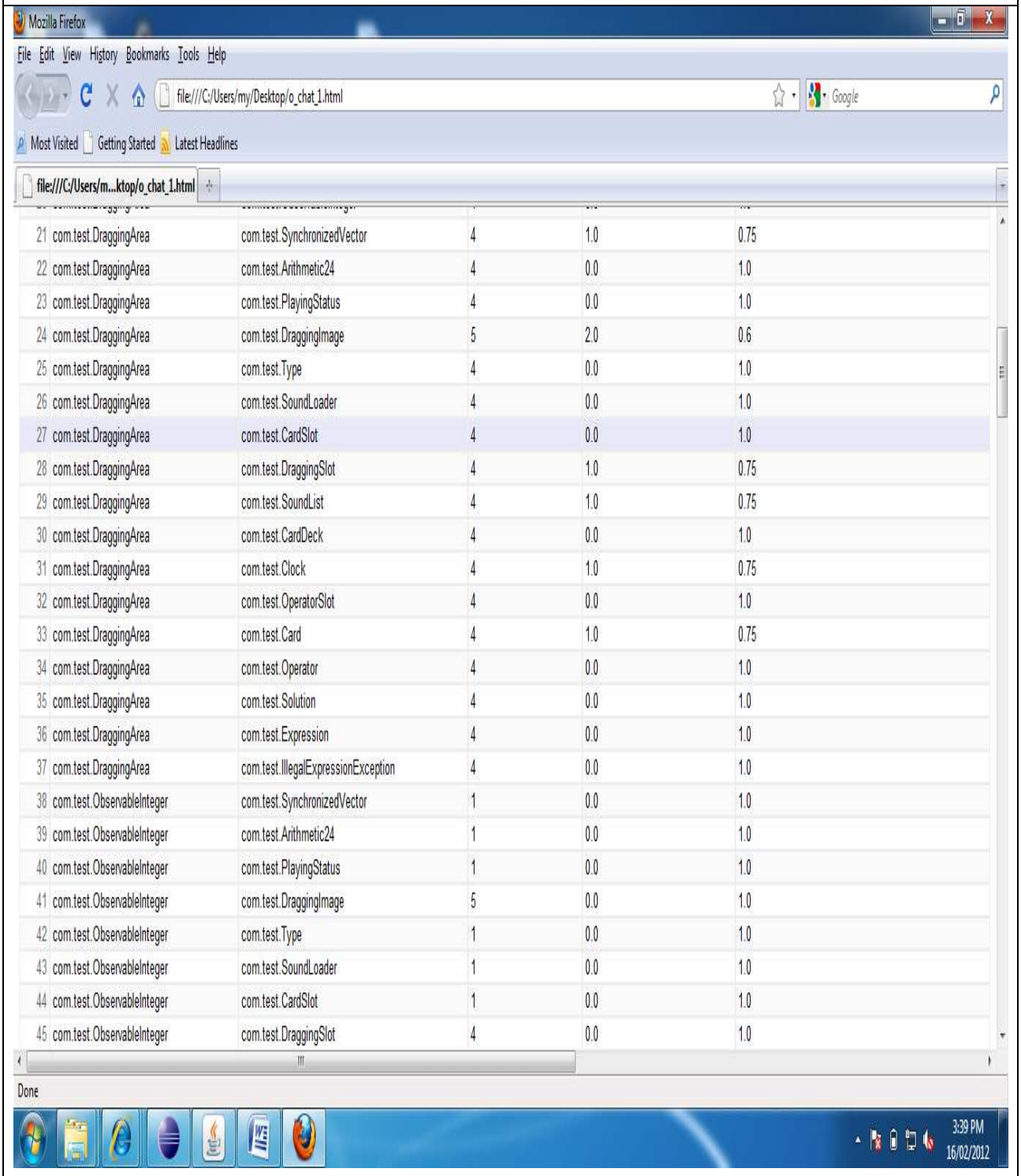
Figure 6. 3: Distance table created using integrated coupling

For cluster level 1



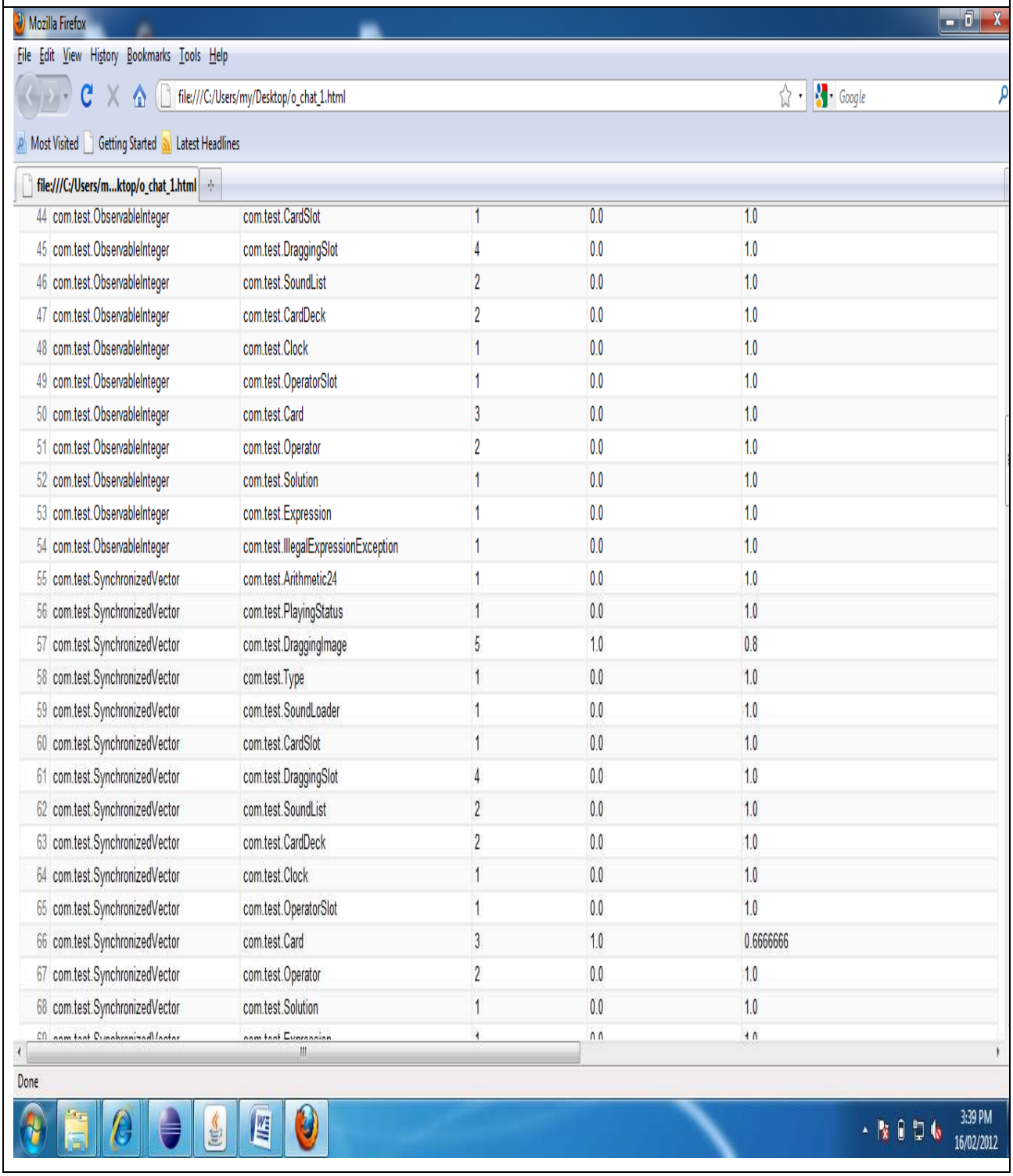
	Source Class Name	Destination Class Name	Union Count	Intersection Count	Distance = 1 - (Intersection/Union)
1	com.test.ScoreKeeper	com.test.DraggingArea	4	1.0	0.75
2	com.test.ScoreKeeper	com.test.ObservableInteger	1	0.0	1.0
3	com.test.ScoreKeeper	com.test.SynchronizedVector	1	0.0	1.0
4	com.test.ScoreKeeper	com.test.Arithmetic24	1	0.0	1.0
5	com.test.ScoreKeeper	com.test.PlayingStatus	1	0.0	1.0
6	com.test.ScoreKeeper	com.test.DraggingImage	5	0.0	1.0
7	com.test.ScoreKeeper	com.test.Type	1	0.0	1.0
8	com.test.ScoreKeeper	com.test.SoundLoader	1	0.0	1.0
9	com.test.ScoreKeeper	com.test.CardSlot	1	0.0	1.0
10	com.test.ScoreKeeper	com.test.DraggingSlot	4	0.0	1.0
11	com.test.ScoreKeeper	com.test.SoundList	2	1.0	0.5
12	com.test.ScoreKeeper	com.test.CardDeck	2	0.0	1.0
13	com.test.ScoreKeeper	com.test.Clock	1	1.0	0.0
14	com.test.ScoreKeeper	com.test.OperatorSlot	1	0.0	1.0
15	com.test.ScoreKeeper	com.test.Card	3	0.0	1.0
16	com.test.ScoreKeeper	com.test.Operator	2	0.0	1.0
17	com.test.ScoreKeeper	com.test.Solution	1	0.0	1.0
18	com.test.ScoreKeeper	com.test.Expression	1	0.0	1.0
19	com.test.ScoreKeeper	com.test.IllegalExpressionException	1	0.0	1.0
20	com.test.DraggingArea	com.test.ObservableInteger	4	0.0	1.0
21	com.test.DraggingArea	com.test.SynchronizedVector	4	1.0	0.75

Figure 6.3 continued…….



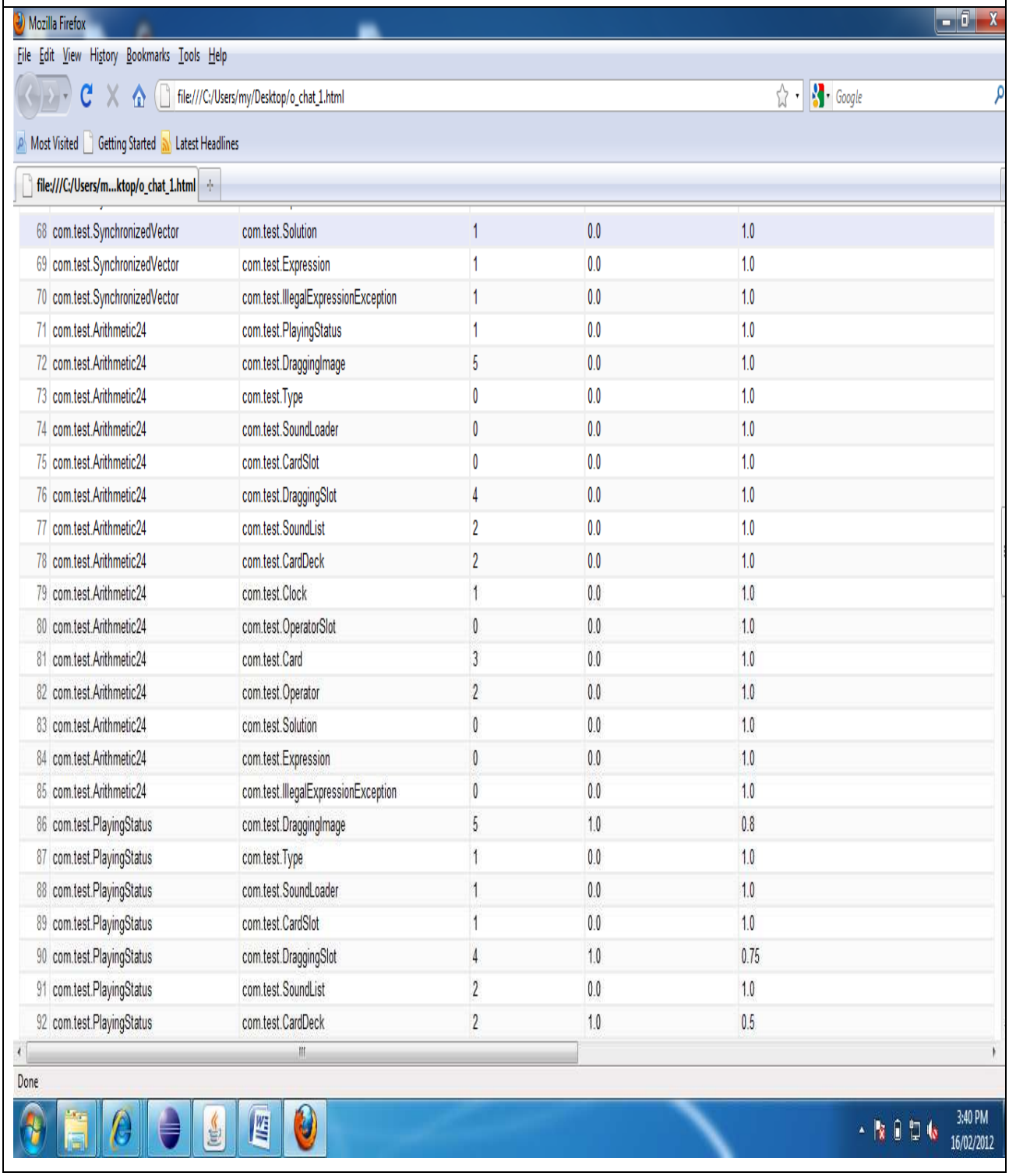
21	com.test.DraggingArea	com.test.SynchronizedVector	4	1.0	0.75
22	com.test.DraggingArea	com.test.Arithmetic24	4	0.0	1.0
23	com.test.DraggingArea	com.test.PlayingStatus	4	0.0	1.0
24	com.test.DraggingArea	com.test.DraggingImage	5	2.0	0.6
25	com.test.DraggingArea	com.test.Type	4	0.0	1.0
26	com.test.DraggingArea	com.test.SoundLoader	4	0.0	1.0
27	com.test.DraggingArea	com.test.CardSlot	4	0.0	1.0
28	com.test.DraggingArea	com.test.DraggingSlot	4	1.0	0.75
29	com.test.DraggingArea	com.test.SoundList	4	1.0	0.75
30	com.test.DraggingArea	com.test.CardDeck	4	0.0	1.0
31	com.test.DraggingArea	com.test.Clock	4	1.0	0.75
32	com.test.DraggingArea	com.test.OperatorSlot	4	0.0	1.0
33	com.test.DraggingArea	com.test.Card	4	1.0	0.75
34	com.test.DraggingArea	com.test.Operator	4	0.0	1.0
35	com.test.DraggingArea	com.test.Solution	4	0.0	1.0
36	com.test.DraggingArea	com.test.Expression	4	0.0	1.0
37	com.test.DraggingArea	com.test.IllegalExpressionException	4	0.0	1.0
38	com.test.ObservableInteger	com.test.SynchronizedVector	1	0.0	1.0
39	com.test.ObservableInteger	com.test.Arithmetic24	1	0.0	1.0
40	com.test.ObservableInteger	com.test.PlayingStatus	1	0.0	1.0
41	com.test.ObservableInteger	com.test.DraggingImage	5	0.0	1.0
42	com.test.ObservableInteger	com.test.Type	1	0.0	1.0
43	com.test.ObservableInteger	com.test.SoundLoader	1	0.0	1.0
44	com.test.ObservableInteger	com.test.CardSlot	1	0.0	1.0
45	com.test.ObservableInteger	com.test.DraggingSlot	4	0.0	1.0

Figure 6.3 continued……..



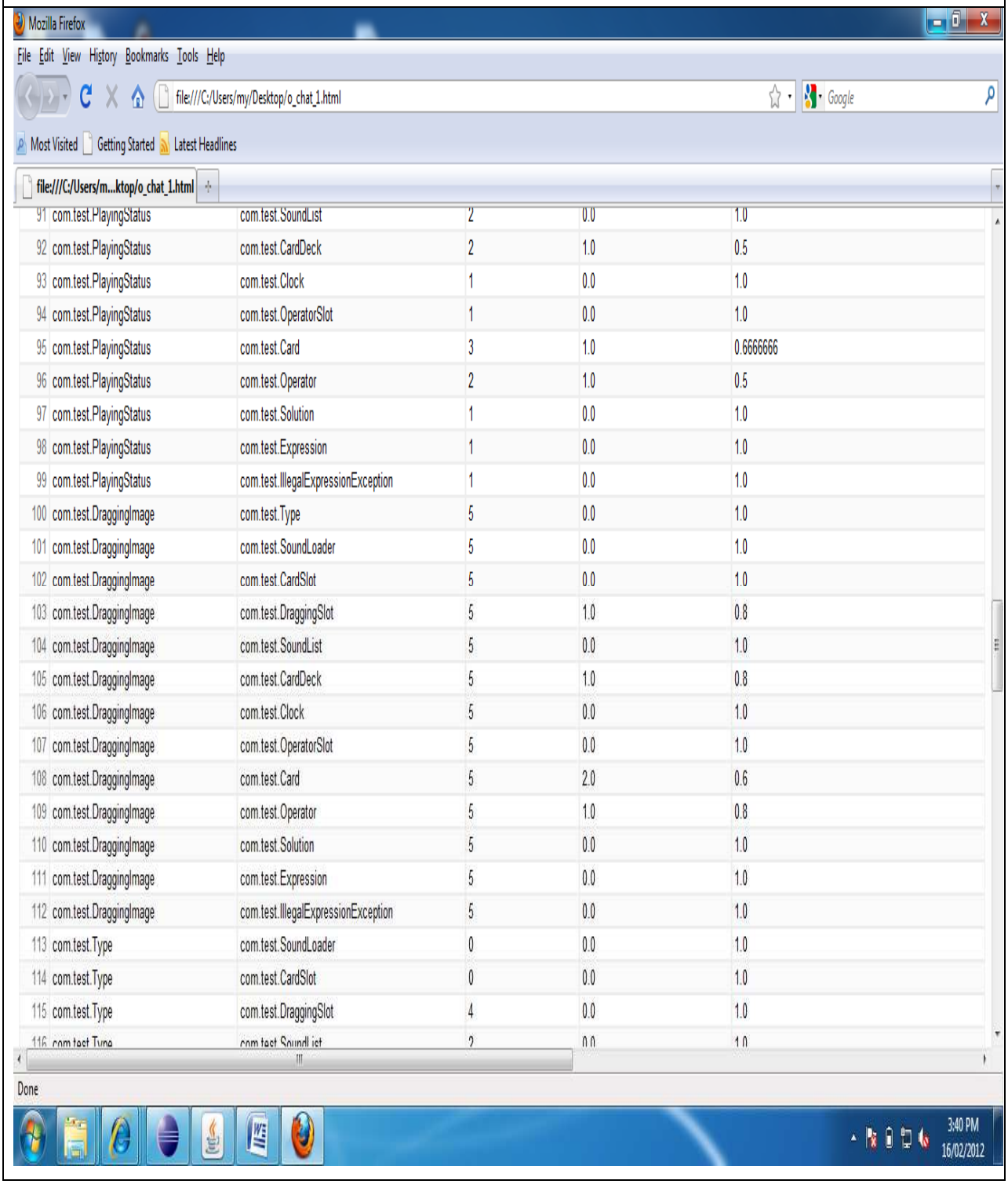
44	com.test.ObservableInteger	com.test.CardSlot	1	0.0	1.0
45	com.test.ObservableInteger	com.test.DraggingSlot	4	0.0	1.0
46	com.test.ObservableInteger	com.test.SoundList	2	0.0	1.0
47	com.test.ObservableInteger	com.test.CardDeck	2	0.0	1.0
48	com.test.ObservableInteger	com.test.Clock	1	0.0	1.0
49	com.test.ObservableInteger	com.test.OperatorSlot	1	0.0	1.0
50	com.test.ObservableInteger	com.test.Card	3	0.0	1.0
51	com.test.ObservableInteger	com.test.Operator	2	0.0	1.0
52	com.test.ObservableInteger	com.test.Solution	1	0.0	1.0
53	com.test.ObservableInteger	com.test.Expression	1	0.0	1.0
54	com.test.ObservableInteger	com.test.IllegalExpressionException	1	0.0	1.0
55	com.test.SynchronizedVector	com.test.Arithmetic24	1	0.0	1.0
56	com.test.SynchronizedVector	com.test.PlayingStatus	1	0.0	1.0
57	com.test.SynchronizedVector	com.test.DraggingImage	5	1.0	0.8
58	com.test.SynchronizedVector	com.test.Type	1	0.0	1.0
59	com.test.SynchronizedVector	com.test.SoundLoader	1	0.0	1.0
60	com.test.SynchronizedVector	com.test.CardSlot	1	0.0	1.0
61	com.test.SynchronizedVector	com.test.DraggingSlot	4	0.0	1.0
62	com.test.SynchronizedVector	com.test.SoundList	2	0.0	1.0
63	com.test.SynchronizedVector	com.test.CardDeck	2	0.0	1.0
64	com.test.SynchronizedVector	com.test.Clock	1	0.0	1.0
65	com.test.SynchronizedVector	com.test.OperatorSlot	1	0.0	1.0
66	com.test.SynchronizedVector	com.test.Card	3	1.0	0.6666666
67	com.test.SynchronizedVector	com.test.Operator	2	0.0	1.0
68	com.test.SynchronizedVector	com.test.Solution	1	0.0	1.0

Figure 6.3 continued……



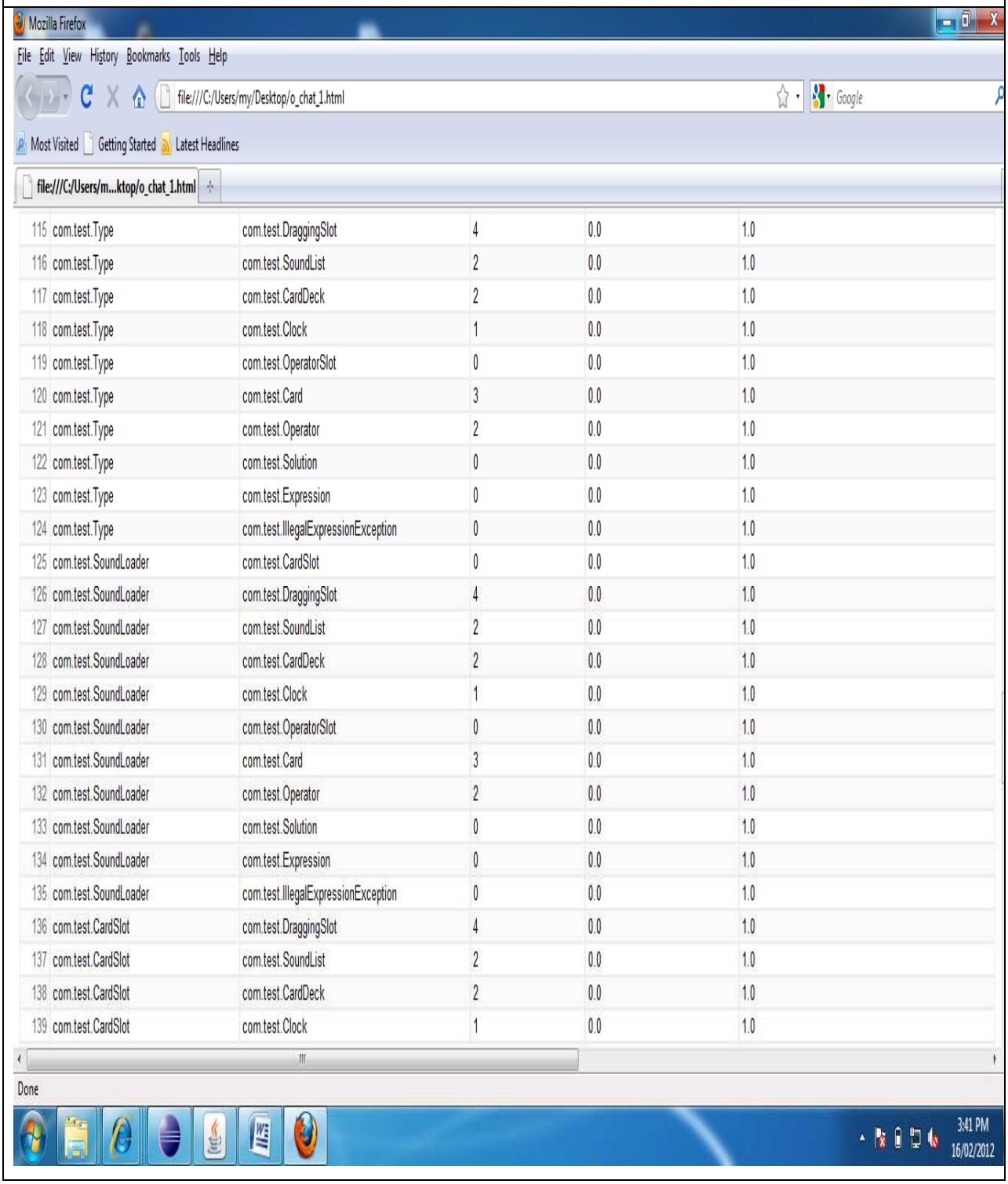
68	com.test.SynchronizedVector	com.test.Solution	1	0.0	1.0
69	com.test.SynchronizedVector	com.test.Expression	1	0.0	1.0
70	com.test.SynchronizedVector	com.test.IllegalExpressionException	1	0.0	1.0
71	com.test.Arithmetic24	com.test.PlayingStatus	1	0.0	1.0
72	com.test.Arithmetic24	com.test.DraggingImage	5	0.0	1.0
73	com.test.Arithmetic24	com.test.Type	0	0.0	1.0
74	com.test.Arithmetic24	com.test.SoundLoader	0	0.0	1.0
75	com.test.Arithmetic24	com.test.CardSlot	0	0.0	1.0
76	com.test.Arithmetic24	com.test.DraggingSlot	4	0.0	1.0
77	com.test.Arithmetic24	com.test.SoundList	2	0.0	1.0
78	com.test.Arithmetic24	com.test.CardDeck	2	0.0	1.0
79	com.test.Arithmetic24	com.test.Clock	1	0.0	1.0
80	com.test.Arithmetic24	com.test.OperatorSlot	0	0.0	1.0
81	com.test.Arithmetic24	com.test.Card	3	0.0	1.0
82	com.test.Arithmetic24	com.test.Operator	2	0.0	1.0
83	com.test.Arithmetic24	com.test.Solution	0	0.0	1.0
84	com.test.Arithmetic24	com.test.Expression	0	0.0	1.0
85	com.test.Arithmetic24	com.test.IllegalExpressionException	0	0.0	1.0
86	com.test.PlayingStatus	com.test.DraggingImage	5	1.0	0.8
87	com.test.PlayingStatus	com.test.Type	1	0.0	1.0
88	com.test.PlayingStatus	com.test.SoundLoader	1	0.0	1.0
89	com.test.PlayingStatus	com.test.CardSlot	1	0.0	1.0
90	com.test.PlayingStatus	com.test.DraggingSlot	4	1.0	0.75
91	com.test.PlayingStatus	com.test.SoundList	2	0.0	1.0
92	com.test.PlayingStatus	com.test.CardDeck	2	1.0	0.5

Figure 6.3 continued…….



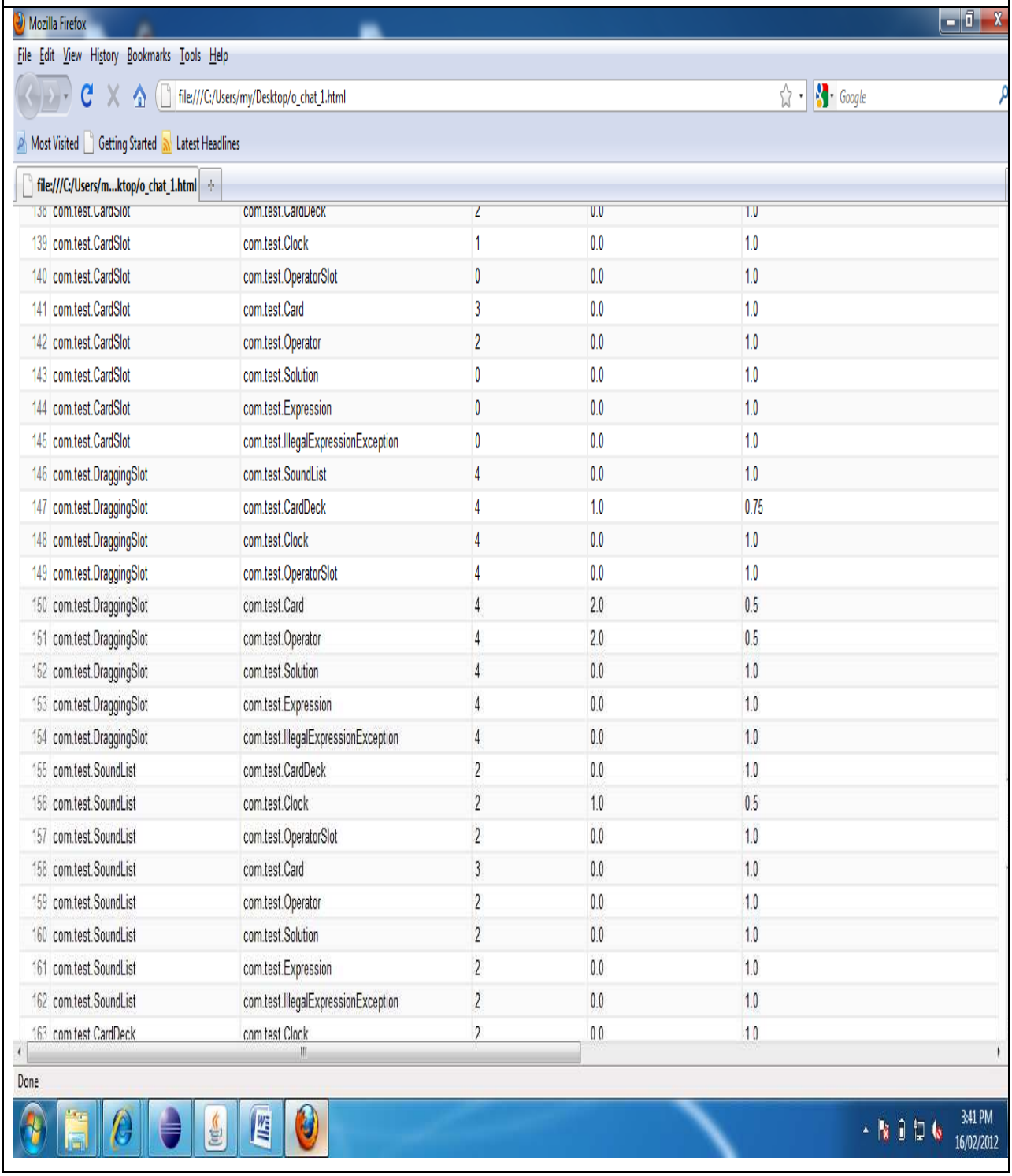
91	com.test.PlayingStatus	com.test.SoundList	2	0.0	1.0
92	com.test.PlayingStatus	com.test.CardDeck	2	1.0	0.5
93	com.test.PlayingStatus	com.test.Clock	1	0.0	1.0
94	com.test.PlayingStatus	com.test.OperatorSlot	1	0.0	1.0
95	com.test.PlayingStatus	com.test.Card	3	1.0	0.6666666
96	com.test.PlayingStatus	com.test.Operator	2	1.0	0.5
97	com.test.PlayingStatus	com.test.Solution	1	0.0	1.0
98	com.test.PlayingStatus	com.test.Expression	1	0.0	1.0
99	com.test.PlayingStatus	com.test.IllegalExpressionException	1	0.0	1.0
100	com.test.DraggingImage	com.test.Type	5	0.0	1.0
101	com.test.DraggingImage	com.test.SoundLoader	5	0.0	1.0
102	com.test.DraggingImage	com.test.CardSlot	5	0.0	1.0
103	com.test.DraggingImage	com.test.DraggingSlot	5	1.0	0.8
104	com.test.DraggingImage	com.test.SoundList	5	0.0	1.0
105	com.test.DraggingImage	com.test.CardDeck	5	1.0	0.8
106	com.test.DraggingImage	com.test.Clock	5	0.0	1.0
107	com.test.DraggingImage	com.test.OperatorSlot	5	0.0	1.0
108	com.test.DraggingImage	com.test.Card	5	2.0	0.6
109	com.test.DraggingImage	com.test.Operator	5	1.0	0.8
110	com.test.DraggingImage	com.test.Solution	5	0.0	1.0
111	com.test.DraggingImage	com.test.Expression	5	0.0	1.0
112	com.test.DraggingImage	com.test.IllegalExpressionException	5	0.0	1.0
113	com.test.Type	com.test.SoundLoader	0	0.0	1.0
114	com.test.Type	com.test.CardSlot	0	0.0	1.0
115	com.test.Type	com.test.DraggingSlot	4	0.0	1.0
116	com.test.Type	com.test.SoundList	2	0.0	1.0

Figure 6.3 continued…….



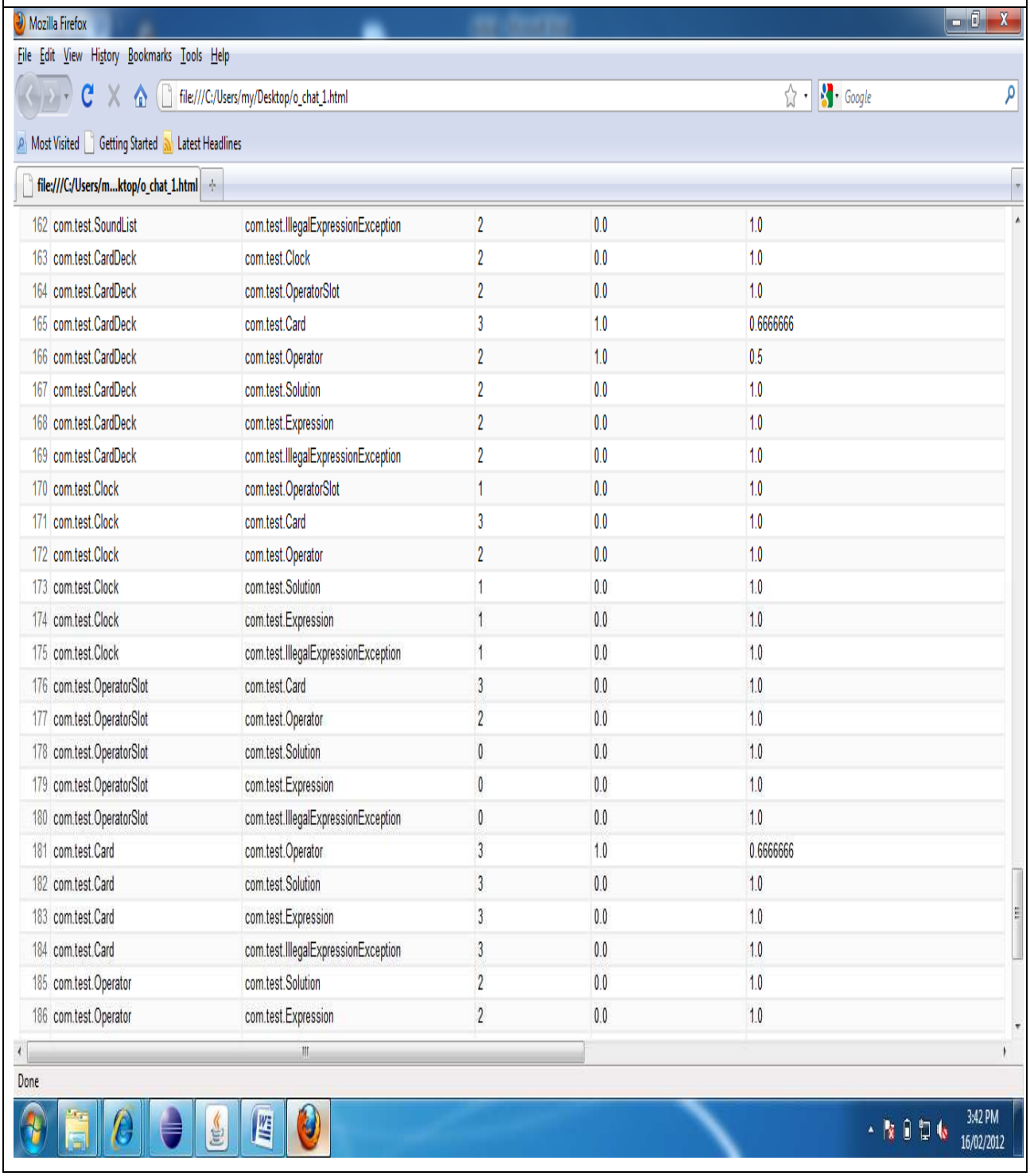
115	com.test.Type	com.test.DraggingSlot	4	0.0	1.0
116	com.test.Type	com.test.SoundList	2	0.0	1.0
117	com.test.Type	com.test.CardDeck	2	0.0	1.0
118	com.test.Type	com.test.Clock	1	0.0	1.0
119	com.test.Type	com.test.OperatorSlot	0	0.0	1.0
120	com.test.Type	com.test.Card	3	0.0	1.0
121	com.test.Type	com.test.Operator	2	0.0	1.0
122	com.test.Type	com.test.Solution	0	0.0	1.0
123	com.test.Type	com.test.Expression	0	0.0	1.0
124	com.test.Type	com.test.IllegalExpressionException	0	0.0	1.0
125	com.test.SoundLoader	com.test.CardSlot	0	0.0	1.0
126	com.test.SoundLoader	com.test.DraggingSlot	4	0.0	1.0
127	com.test.SoundLoader	com.test.SoundList	2	0.0	1.0
128	com.test.SoundLoader	com.test.CardDeck	2	0.0	1.0
129	com.test.SoundLoader	com.test.Clock	1	0.0	1.0
130	com.test.SoundLoader	com.test.OperatorSlot	0	0.0	1.0
131	com.test.SoundLoader	com.test.Card	3	0.0	1.0
132	com.test.SoundLoader	com.test.Operator	2	0.0	1.0
133	com.test.SoundLoader	com.test.Solution	0	0.0	1.0
134	com.test.SoundLoader	com.test.Expression	0	0.0	1.0
135	com.test.SoundLoader	com.test.IllegalExpressionException	0	0.0	1.0
136	com.test.CardSlot	com.test.DraggingSlot	4	0.0	1.0
137	com.test.CardSlot	com.test.SoundList	2	0.0	1.0
138	com.test.CardSlot	com.test.CardDeck	2	0.0	1.0
139	com.test.CardSlot	com.test.Clock	1	0.0	1.0

Figure 6.3 continued…….



138	com.test.CardSlot	com.test.CardDeck	2	0.0	1.0
139	com.test.CardSlot	com.test.Clock	1	0.0	1.0
140	com.test.CardSlot	com.test.OperatorSlot	0	0.0	1.0
141	com.test.CardSlot	com.test.Card	3	0.0	1.0
142	com.test.CardSlot	com.test.Operator	2	0.0	1.0
143	com.test.CardSlot	com.test.Solution	0	0.0	1.0
144	com.test.CardSlot	com.test.Expression	0	0.0	1.0
145	com.test.CardSlot	com.test.IllegalExpressionException	0	0.0	1.0
146	com.test.DraggingSlot	com.test.SoundList	4	0.0	1.0
147	com.test.DraggingSlot	com.test.CardDeck	4	1.0	0.75
148	com.test.DraggingSlot	com.test.Clock	4	0.0	1.0
149	com.test.DraggingSlot	com.test.OperatorSlot	4	0.0	1.0
150	com.test.DraggingSlot	com.test.Card	4	2.0	0.5
151	com.test.DraggingSlot	com.test.Operator	4	2.0	0.5
152	com.test.DraggingSlot	com.test.Solution	4	0.0	1.0
153	com.test.DraggingSlot	com.test.Expression	4	0.0	1.0
154	com.test.DraggingSlot	com.test.IllegalExpressionException	4	0.0	1.0
155	com.test.SoundList	com.test.CardDeck	2	0.0	1.0
156	com.test.SoundList	com.test.Clock	2	1.0	0.5
157	com.test.SoundList	com.test.OperatorSlot	2	0.0	1.0
158	com.test.SoundList	com.test.Card	3	0.0	1.0
159	com.test.SoundList	com.test.Operator	2	0.0	1.0
160	com.test.SoundList	com.test.Solution	2	0.0	1.0
161	com.test.SoundList	com.test.Expression	2	0.0	1.0
162	com.test.SoundList	com.test.IllegalExpressionException	2	0.0	1.0
163	com.test.CardDeck	com.test.Clock	2	0.0	1.0

Figure 6.3 continued…….



162	com.test.SoundList	com.test.IllegalExpressionException	2	0.0	1.0
163	com.test.CardDeck	com.test.Clock	2	0.0	1.0
164	com.test.CardDeck	com.test.OperatorSlot	2	0.0	1.0
165	com.test.CardDeck	com.test.Card	3	1.0	0.6666666
166	com.test.CardDeck	com.test.Operator	2	1.0	0.5
167	com.test.CardDeck	com.test.Solution	2	0.0	1.0
168	com.test.CardDeck	com.test.Expression	2	0.0	1.0
169	com.test.CardDeck	com.test.IllegalExpressionException	2	0.0	1.0
170	com.test.Clock	com.test.OperatorSlot	1	0.0	1.0
171	com.test.Clock	com.test.Card	3	0.0	1.0
172	com.test.Clock	com.test.Operator	2	0.0	1.0
173	com.test.Clock	com.test.Solution	1	0.0	1.0
174	com.test.Clock	com.test.Expression	1	0.0	1.0
175	com.test.Clock	com.test.IllegalExpressionException	1	0.0	1.0
176	com.test.OperatorSlot	com.test.Card	3	0.0	1.0
177	com.test.OperatorSlot	com.test.Operator	2	0.0	1.0
178	com.test.OperatorSlot	com.test.Solution	0	0.0	1.0
179	com.test.OperatorSlot	com.test.Expression	0	0.0	1.0
180	com.test.OperatorSlot	com.test.IllegalExpressionException	0	0.0	1.0
181	com.test.Card	com.test.Operator	3	1.0	0.6666666
182	com.test.Card	com.test.Solution	3	0.0	1.0
183	com.test.Card	com.test.Expression	3	0.0	1.0
184	com.test.Card	com.test.IllegalExpressionException	3	0.0	1.0
185	com.test.Operator	com.test.Solution	2	0.0	1.0
186	com.test.Operator	com.test.Expression	2	0.0	1.0

Figure 6.3 continued…….

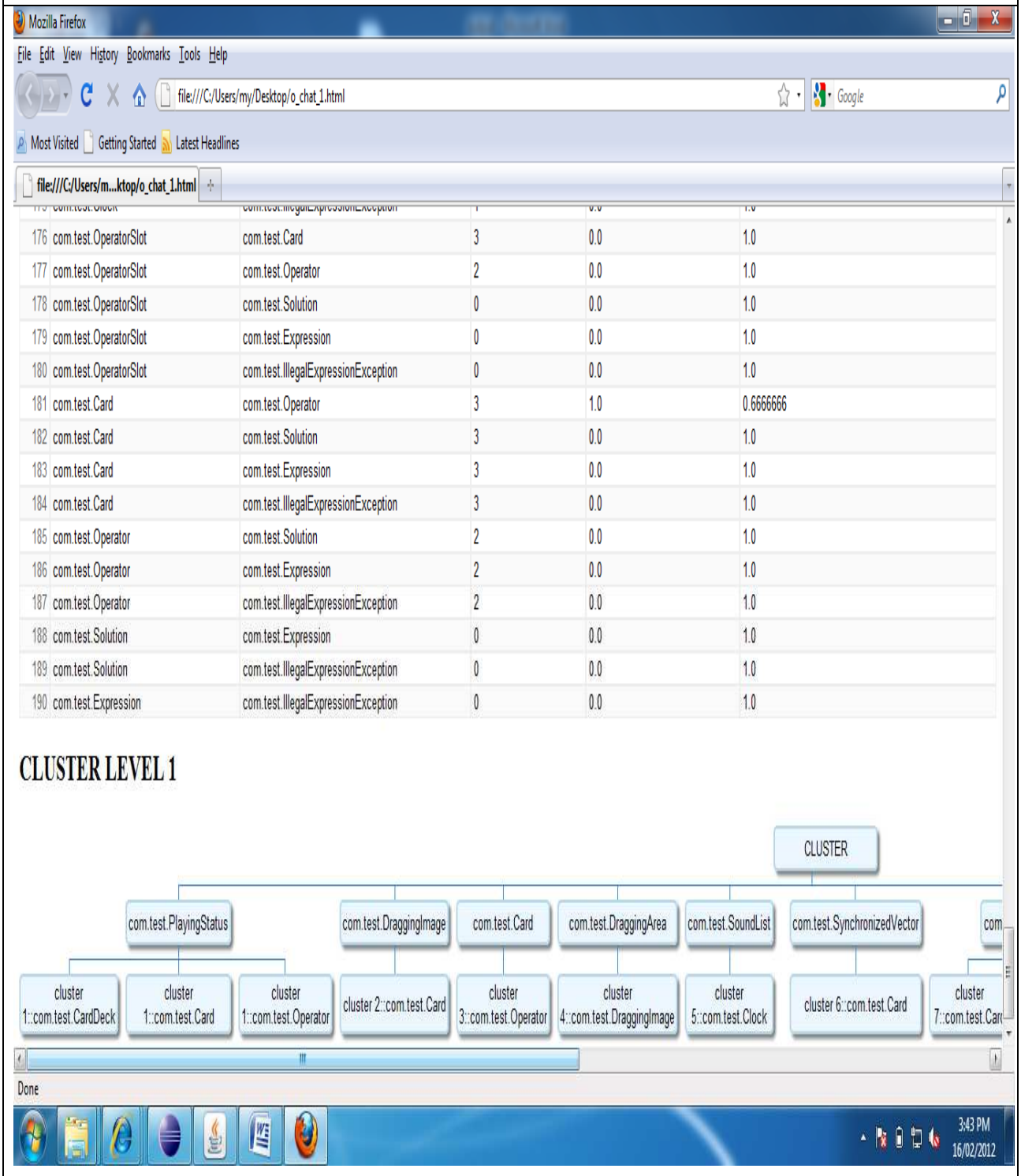
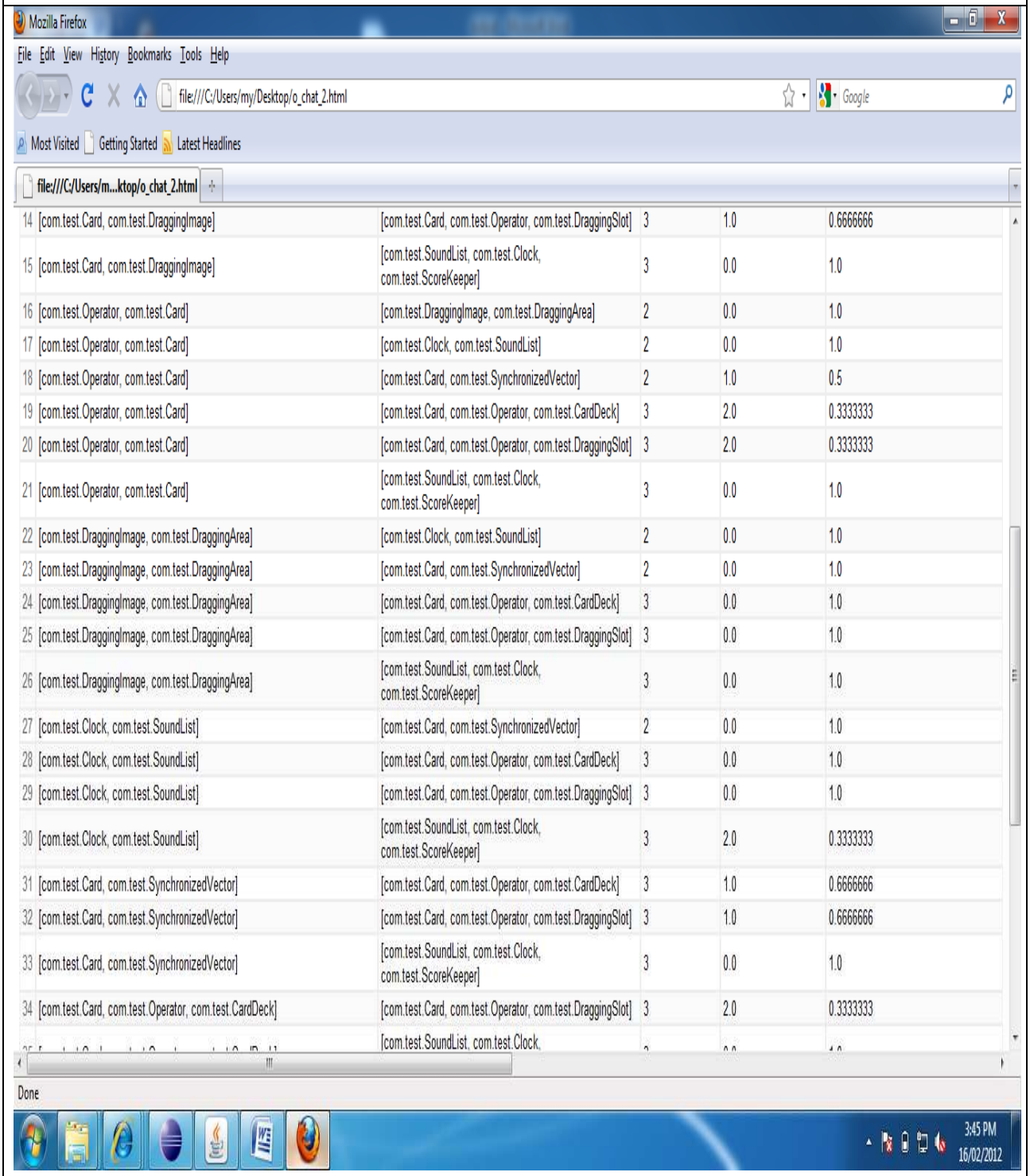


Figure 6.3 continued…….



Mozilla Firefox

File Edit View History Bookmarks Tools Help

file:///C:/Users/my/Desktop/o_chat_2.html

Most Visited Getting Started Latest Headlines

file:///C:/Users/m...ktop/o_chat_2.html

14	[com.test.Card, com.test.DraggingImage]	[com.test.Card, com.test.Operator, com.test.DraggingSlot]	3	1.0	0.6666666
15	[com.test.Card, com.test.DraggingImage]	[com.test.SoundList, com.test.Clock, com.test.ScoreKeeper]	3	0.0	1.0
16	[com.test.Operator, com.test.Card]	[com.test.DraggingImage, com.test.DraggingArea]	2	0.0	1.0
17	[com.test.Operator, com.test.Card]	[com.test.Clock, com.test.SoundList]	2	0.0	1.0
18	[com.test.Operator, com.test.Card]	[com.test.Card, com.test.SynchronizedVector]	2	1.0	0.5
19	[com.test.Operator, com.test.Card]	[com.test.Card, com.test.Operator, com.test.CardDeck]	3	2.0	0.3333333
20	[com.test.Operator, com.test.Card]	[com.test.Card, com.test.Operator, com.test.DraggingSlot]	3	2.0	0.3333333
21	[com.test.Operator, com.test.Card]	[com.test.SoundList, com.test.Clock, com.test.ScoreKeeper]	3	0.0	1.0
22	[com.test.DraggingImage, com.test.DraggingArea]	[com.test.Clock, com.test.SoundList]	2	0.0	1.0
23	[com.test.DraggingImage, com.test.DraggingArea]	[com.test.Card, com.test.SynchronizedVector]	2	0.0	1.0
24	[com.test.DraggingImage, com.test.DraggingArea]	[com.test.Card, com.test.Operator, com.test.CardDeck]	3	0.0	1.0
25	[com.test.DraggingImage, com.test.DraggingArea]	[com.test.Card, com.test.Operator, com.test.DraggingSlot]	3	0.0	1.0
26	[com.test.DraggingImage, com.test.DraggingArea]	[com.test.SoundList, com.test.Clock, com.test.ScoreKeeper]	3	0.0	1.0
27	[com.test.Clock, com.test.SoundList]	[com.test.Card, com.test.SynchronizedVector]	2	0.0	1.0
28	[com.test.Clock, com.test.SoundList]	[com.test.Card, com.test.Operator, com.test.CardDeck]	3	0.0	1.0
29	[com.test.Clock, com.test.SoundList]	[com.test.Card, com.test.Operator, com.test.DraggingSlot]	3	0.0	1.0
30	[com.test.Clock, com.test.SoundList]	[com.test.SoundList, com.test.Clock, com.test.ScoreKeeper]	3	2.0	0.3333333
31	[com.test.Card, com.test.SynchronizedVector]	[com.test.Card, com.test.Operator, com.test.CardDeck]	3	1.0	0.6666666
32	[com.test.Card, com.test.SynchronizedVector]	[com.test.Card, com.test.Operator, com.test.DraggingSlot]	3	1.0	0.6666666
33	[com.test.Card, com.test.SynchronizedVector]	[com.test.SoundList, com.test.Clock, com.test.ScoreKeeper]	3	0.0	1.0
34	[com.test.Card, com.test.Operator, com.test.CardDeck]	[com.test.Card, com.test.Operator, com.test.DraggingSlot]	3	2.0	0.3333333
		[com.test.SoundList, com.test.Clock,			

Done

3:45 PM
16/02/2012

Figure 6.3 continued…….

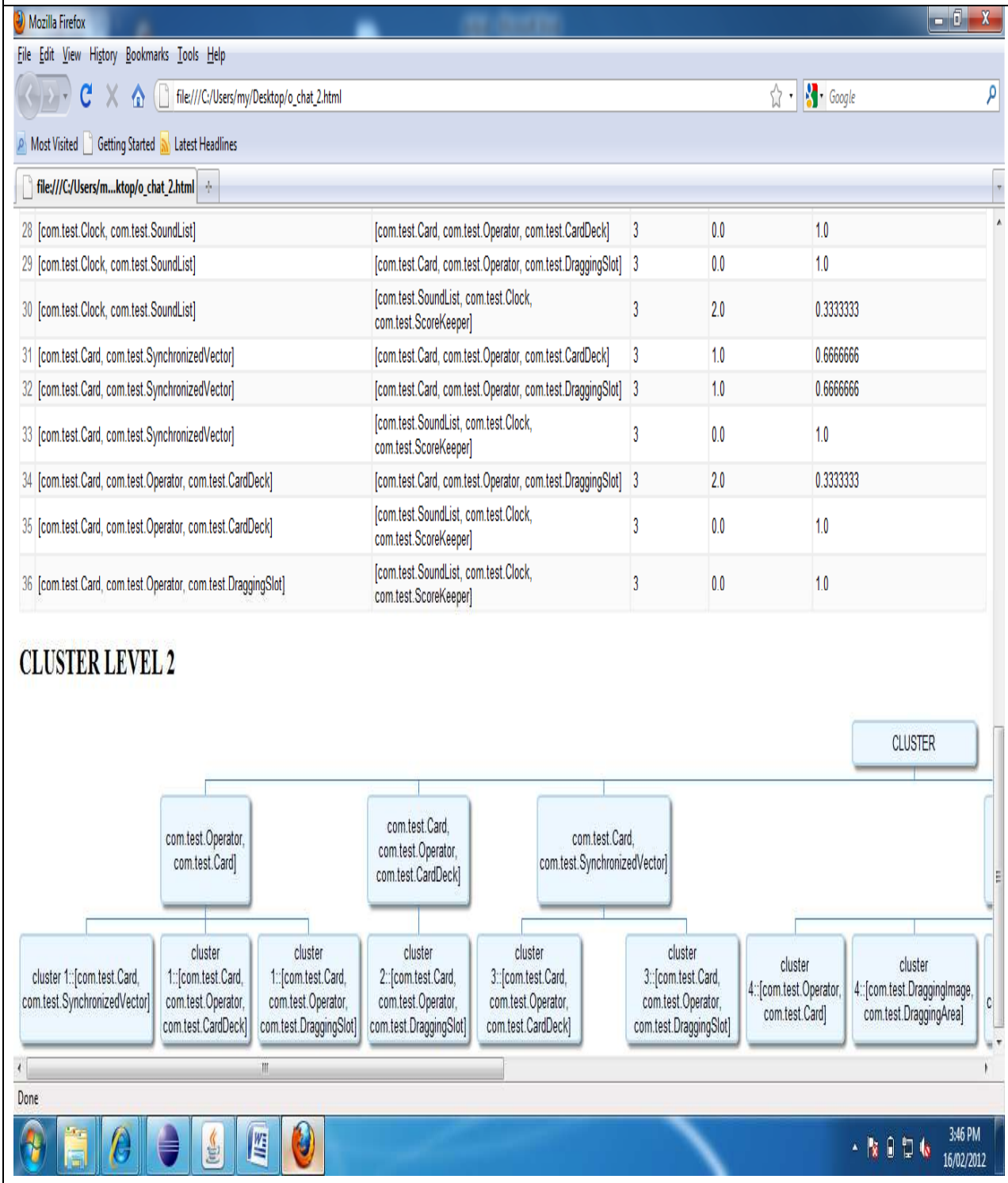
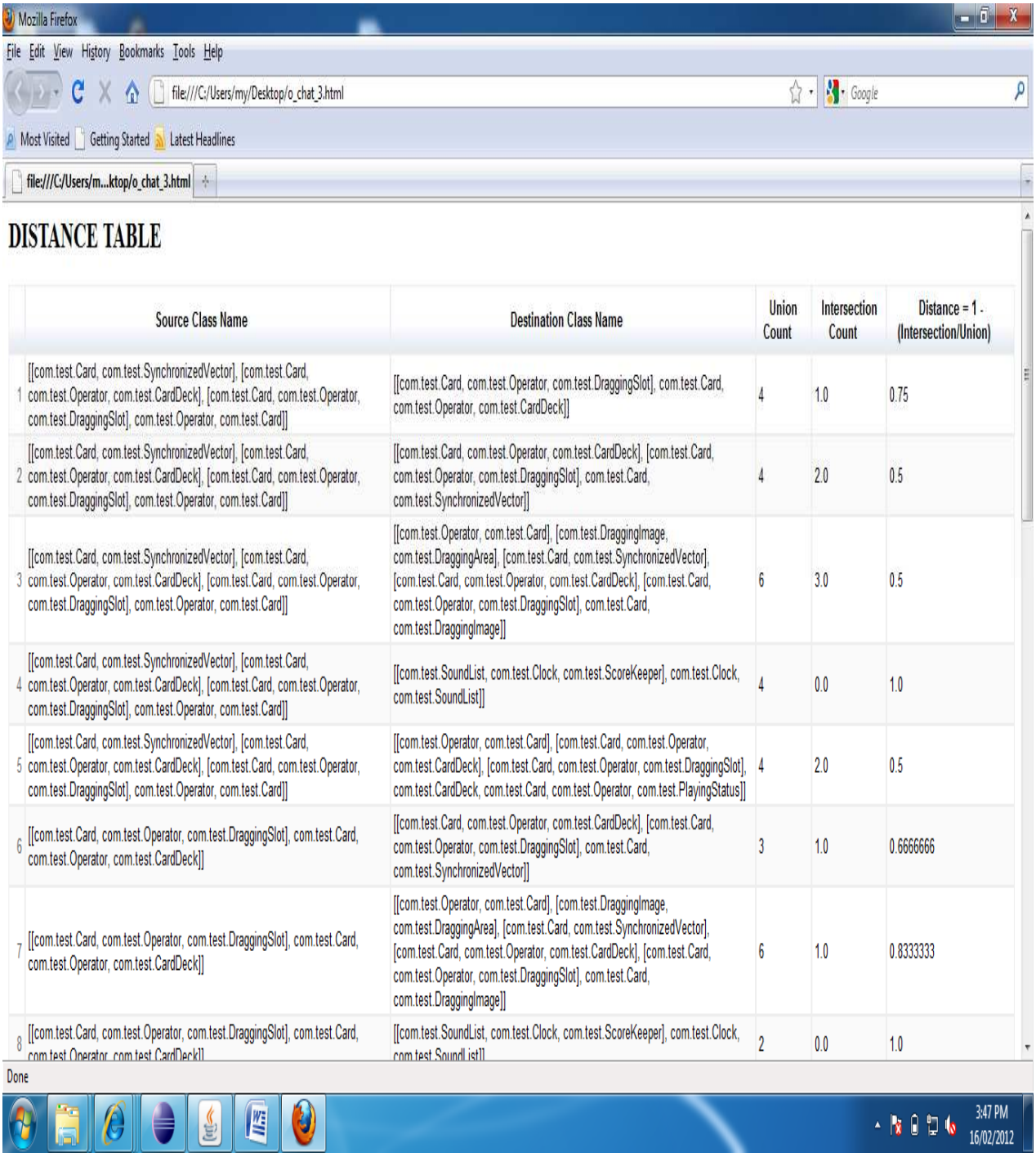


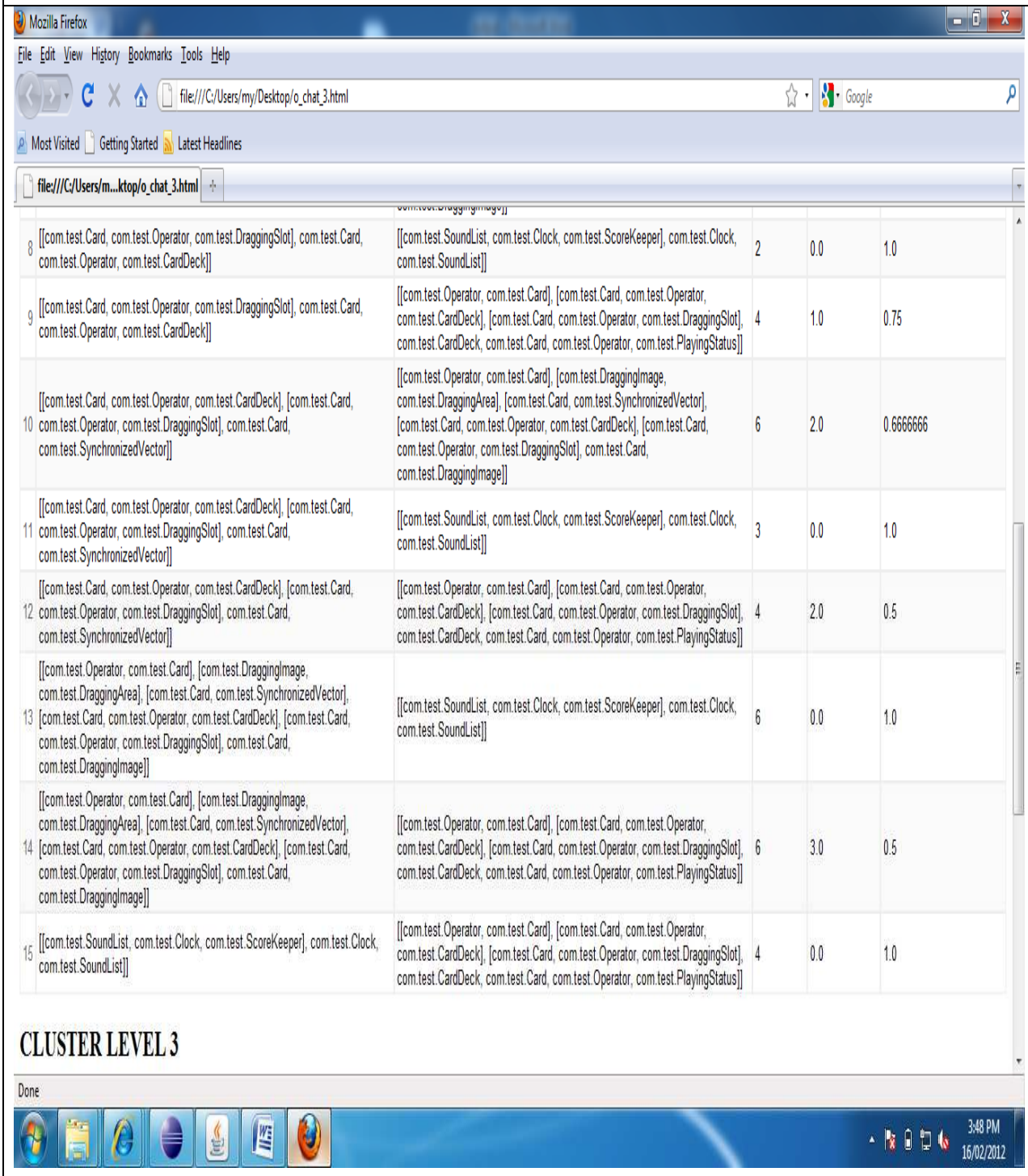
Figure 6.3 continued…….

Cluster level 3



	Source Class Name	Destination Class Name	Union Count	Intersection Count	Distance = 1 - (Intersection/Union)
1	[[com.test.Card, com.test.SynchronizedVector], [com.test.Card, com.test.Operator, com.test.CardDeck], [com.test.Card, com.test.Operator, com.test.DraggingSlot], com.test.Operator, com.test.Card]]	[[com.test.Card, com.test.Operator, com.test.DraggingSlot], com.test.Card, com.test.Operator, com.test.CardDeck]]	4	1.0	0.75
2	[[com.test.Card, com.test.SynchronizedVector], [com.test.Card, com.test.Operator, com.test.CardDeck], [com.test.Card, com.test.Operator, com.test.DraggingSlot], com.test.Operator, com.test.Card]]	[[com.test.Card, com.test.Operator, com.test.CardDeck], [com.test.Card, com.test.Operator, com.test.DraggingSlot], com.test.Card, com.test.SynchronizedVector]]	4	2.0	0.5
3	[[com.test.Card, com.test.SynchronizedVector], [com.test.Card, com.test.Operator, com.test.CardDeck], [com.test.Card, com.test.Operator, com.test.DraggingSlot], com.test.Operator, com.test.Card]]	[[com.test.Operator, com.test.Card], [com.test.DraggingImage, com.test.DraggingArea], [com.test.Card, com.test.SynchronizedVector], [com.test.Card, com.test.Operator, com.test.CardDeck], [com.test.Card, com.test.Operator, com.test.DraggingSlot], com.test.Card, com.test.DraggingImage]]	6	3.0	0.5
4	[[com.test.Card, com.test.SynchronizedVector], [com.test.Card, com.test.Operator, com.test.CardDeck], [com.test.Card, com.test.Operator, com.test.DraggingSlot], com.test.Operator, com.test.Card]]	[[com.test.SoundList, com.test.Clock, com.test.ScoreKeeper], com.test.Clock, com.test.SoundList]]	4	0.0	1.0
5	[[com.test.Card, com.test.SynchronizedVector], [com.test.Card, com.test.Operator, com.test.CardDeck], [com.test.Card, com.test.Operator, com.test.DraggingSlot], com.test.Operator, com.test.Card]]	[[com.test.Operator, com.test.Card], [com.test.Card, com.test.Operator, com.test.CardDeck], [com.test.Card, com.test.Operator, com.test.DraggingSlot], com.test.CardDeck, com.test.Card, com.test.Operator, com.test.PlayingStatus]]	4	2.0	0.5
6	[[com.test.Card, com.test.Operator, com.test.DraggingSlot], com.test.Card, com.test.Operator, com.test.CardDeck]]	[[com.test.Card, com.test.Operator, com.test.CardDeck], [com.test.Card, com.test.Operator, com.test.DraggingSlot], com.test.Card, com.test.SynchronizedVector]]	3	1.0	0.6666666
7	[[com.test.Card, com.test.Operator, com.test.DraggingSlot], com.test.Card, com.test.Operator, com.test.CardDeck]]	[[com.test.Operator, com.test.Card], [com.test.DraggingImage, com.test.DraggingArea], [com.test.Card, com.test.SynchronizedVector], [com.test.Card, com.test.Operator, com.test.CardDeck], [com.test.Card, com.test.Operator, com.test.DraggingSlot], com.test.Card, com.test.DraggingImage]]	6	1.0	0.8333333
8	[[com.test.Card, com.test.Operator, com.test.DraggingSlot], com.test.Card, com.test.Operator, com.test.CardDeck]]	[[com.test.SoundList, com.test.Clock, com.test.ScoreKeeper], com.test.Clock, com.test.SoundList]]	2	0.0	1.0

Figure 6.3 continued…….



8	[[com.test.Card, com.test.Operator, com.test.DraggingSlot], com.test.Card, com.test.Operator, com.test.CardDeck]]	[[com.test.SoundList, com.test.Clock, com.test.ScoreKeeper], com.test.Clock, com.test.SoundList]]	2	0.0	1.0
9	[[com.test.Card, com.test.Operator, com.test.DraggingSlot], com.test.Card, com.test.Operator, com.test.CardDeck]]	[[com.test.Operator, com.test.Card], [com.test.Card, com.test.Operator, com.test.CardDeck], [com.test.Card, com.test.Operator, com.test.DraggingSlot], com.test.CardDeck, com.test.Card, com.test.Operator, com.test.PlayingStatus]]	4	1.0	0.75
10	[[com.test.Card, com.test.Operator, com.test.CardDeck], [com.test.Card, com.test.Operator, com.test.DraggingSlot], com.test.Card, com.test.SynchronizedVector]]	[[com.test.Operator, com.test.Card], [com.test.DraggingImage, com.test.DraggingArea], [com.test.Card, com.test.SynchronizedVector], [com.test.Card, com.test.Operator, com.test.CardDeck], [com.test.Card, com.test.Operator, com.test.DraggingSlot], com.test.Card, com.test.DraggingImage]]	6	2.0	0.6666666
11	[[com.test.Card, com.test.Operator, com.test.CardDeck], [com.test.Card, com.test.Operator, com.test.DraggingSlot], com.test.Card, com.test.SynchronizedVector]]	[[com.test.SoundList, com.test.Clock, com.test.ScoreKeeper], com.test.Clock, com.test.SoundList]]	3	0.0	1.0
12	[[com.test.Card, com.test.Operator, com.test.CardDeck], [com.test.Card, com.test.Operator, com.test.DraggingSlot], com.test.Card, com.test.SynchronizedVector]]	[[com.test.Operator, com.test.Card], [com.test.Card, com.test.Operator, com.test.CardDeck], [com.test.Card, com.test.Operator, com.test.DraggingSlot], com.test.CardDeck, com.test.Card, com.test.Operator, com.test.PlayingStatus]]	4	2.0	0.5
13	[[com.test.Operator, com.test.Card], [com.test.DraggingImage, com.test.DraggingArea], [com.test.Card, com.test.SynchronizedVector], [com.test.Card, com.test.Operator, com.test.CardDeck], [com.test.Card, com.test.Operator, com.test.DraggingSlot], com.test.Card, com.test.DraggingImage]]	[[com.test.SoundList, com.test.Clock, com.test.ScoreKeeper], com.test.Clock, com.test.SoundList]]	6	0.0	1.0
14	[[com.test.Operator, com.test.Card], [com.test.DraggingImage, com.test.DraggingArea], [com.test.Card, com.test.SynchronizedVector], [com.test.Card, com.test.Operator, com.test.CardDeck], [com.test.Card, com.test.Operator, com.test.DraggingSlot], com.test.Card, com.test.DraggingImage]]	[[com.test.Operator, com.test.Card], [com.test.Card, com.test.Operator, com.test.CardDeck], [com.test.Card, com.test.Operator, com.test.DraggingSlot], com.test.CardDeck, com.test.Card, com.test.Operator, com.test.PlayingStatus]]	6	3.0	0.5
15	[[com.test.SoundList, com.test.Clock, com.test.ScoreKeeper], com.test.Clock, com.test.SoundList]]	[[com.test.Operator, com.test.Card], [com.test.Card, com.test.Operator, com.test.CardDeck], [com.test.Card, com.test.Operator, com.test.DraggingSlot], com.test.CardDeck, com.test.Card, com.test.Operator, com.test.PlayingStatus]]	4	0.0	1.0

Figure 6.3 continued…….

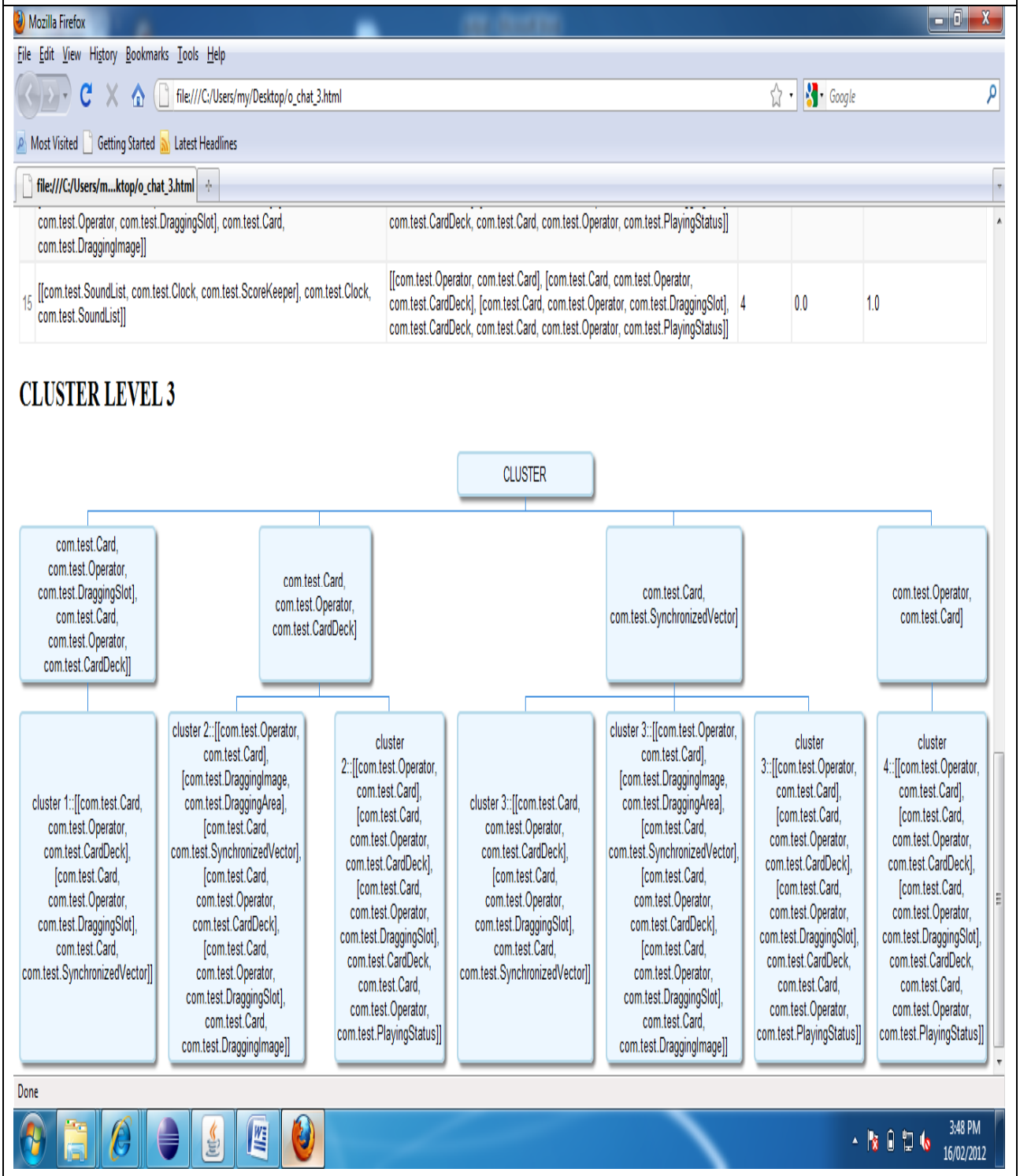
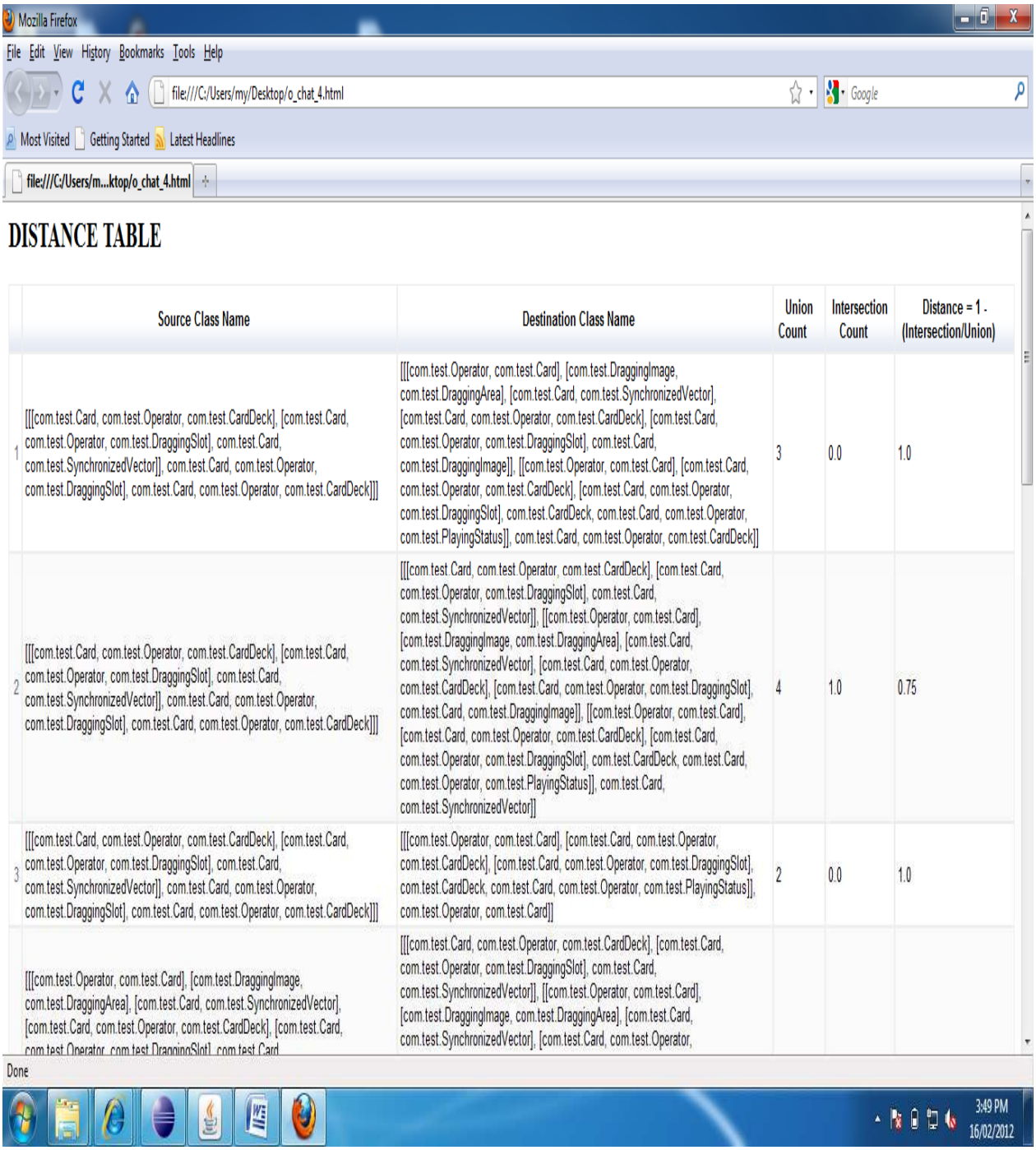


Figure 6.3 continued…….

Cluster level 4



	Source Class Name	Destination Class Name	Union Count	Intersection Count	Distance = 1 - (Intersection/Union)
1	[[[com.test.Card, com.test.Operator, com.test.CardDeck], [com.test.Card, com.test.Operator, com.test.DraggingSlot], com.test.Card, com.test.SynchronizedVector]], com.test.Card, com.test.Operator, com.test.DraggingSlot], com.test.Card, com.test.Operator, com.test.CardDeck]]]	[[[com.test.Operator, com.test.Card], [com.test.DraggingImage, com.test.DraggingArea], [com.test.Card, com.test.SynchronizedVector], [com.test.Card, com.test.Operator, com.test.CardDeck], [com.test.Card, com.test.Operator, com.test.DraggingSlot], com.test.Card, com.test.DraggingImage]], [[com.test.Operator, com.test.Card], [com.test.Card, com.test.Operator, com.test.CardDeck], [com.test.Card, com.test.Operator, com.test.DraggingSlot], com.test.CardDeck, com.test.Card, com.test.Operator, com.test.PlayingStatus]], com.test.Card, com.test.Operator, com.test.CardDeck]]	3	0.0	1.0
2	[[[com.test.Card, com.test.Operator, com.test.CardDeck], [com.test.Card, com.test.Operator, com.test.DraggingSlot], com.test.Card, com.test.SynchronizedVector]], com.test.Card, com.test.Operator, com.test.DraggingSlot], com.test.Card, com.test.Operator, com.test.CardDeck]]]	[[[com.test.Card, com.test.Operator, com.test.CardDeck], [com.test.Card, com.test.Operator, com.test.DraggingSlot], com.test.Card, com.test.SynchronizedVector]], [[com.test.Operator, com.test.Card], [com.test.DraggingImage, com.test.DraggingArea], [com.test.Card, com.test.SynchronizedVector], [com.test.Card, com.test.Operator, com.test.CardDeck], [com.test.Card, com.test.Operator, com.test.DraggingSlot], com.test.Card, com.test.DraggingImage]], [[com.test.Operator, com.test.Card], [com.test.Card, com.test.Operator, com.test.CardDeck], [com.test.Card, com.test.Operator, com.test.DraggingSlot], com.test.CardDeck, com.test.Card, com.test.Operator, com.test.PlayingStatus]], com.test.Card, com.test.SynchronizedVector]]	4	1.0	0.75
3	[[[com.test.Card, com.test.Operator, com.test.CardDeck], [com.test.Card, com.test.Operator, com.test.DraggingSlot], com.test.Card, com.test.SynchronizedVector]], com.test.Card, com.test.Operator, com.test.DraggingSlot], com.test.Card, com.test.Operator, com.test.CardDeck]]]	[[[com.test.Operator, com.test.Card], [com.test.Card, com.test.Operator, com.test.CardDeck], [com.test.Card, com.test.Operator, com.test.DraggingSlot], com.test.CardDeck, com.test.Card, com.test.Operator, com.test.PlayingStatus]], com.test.Operator, com.test.Card]]	2	0.0	1.0
	[[[com.test.Operator, com.test.Card], [com.test.DraggingImage, com.test.DraggingArea], [com.test.Card, com.test.SynchronizedVector], [com.test.Card, com.test.Operator, com.test.CardDeck], [com.test.Card, com.test.Operator, com.test.DraggingSlot], com.test.Card	[[[com.test.Card, com.test.Operator, com.test.CardDeck], [com.test.Card, com.test.Operator, com.test.DraggingSlot], com.test.Card, com.test.SynchronizedVector]], [[com.test.Operator, com.test.Card], [com.test.DraggingImage, com.test.DraggingArea], [com.test.Card, com.test.SynchronizedVector], [com.test.Card, com.test.Operator,			

Figure 6.3 continued…….

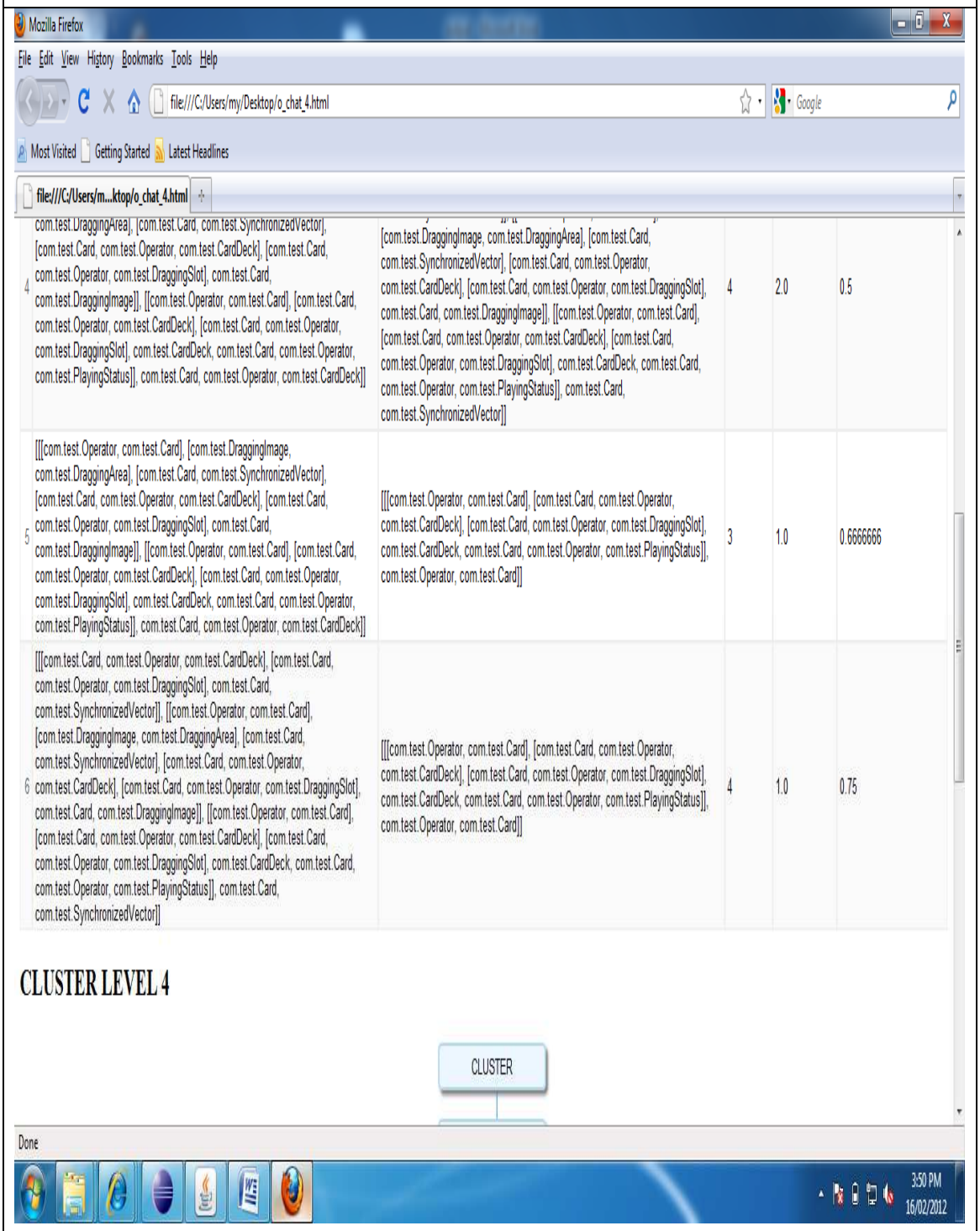


Figure 6.4 : Cluster levels created for 'Arithmetic24' game

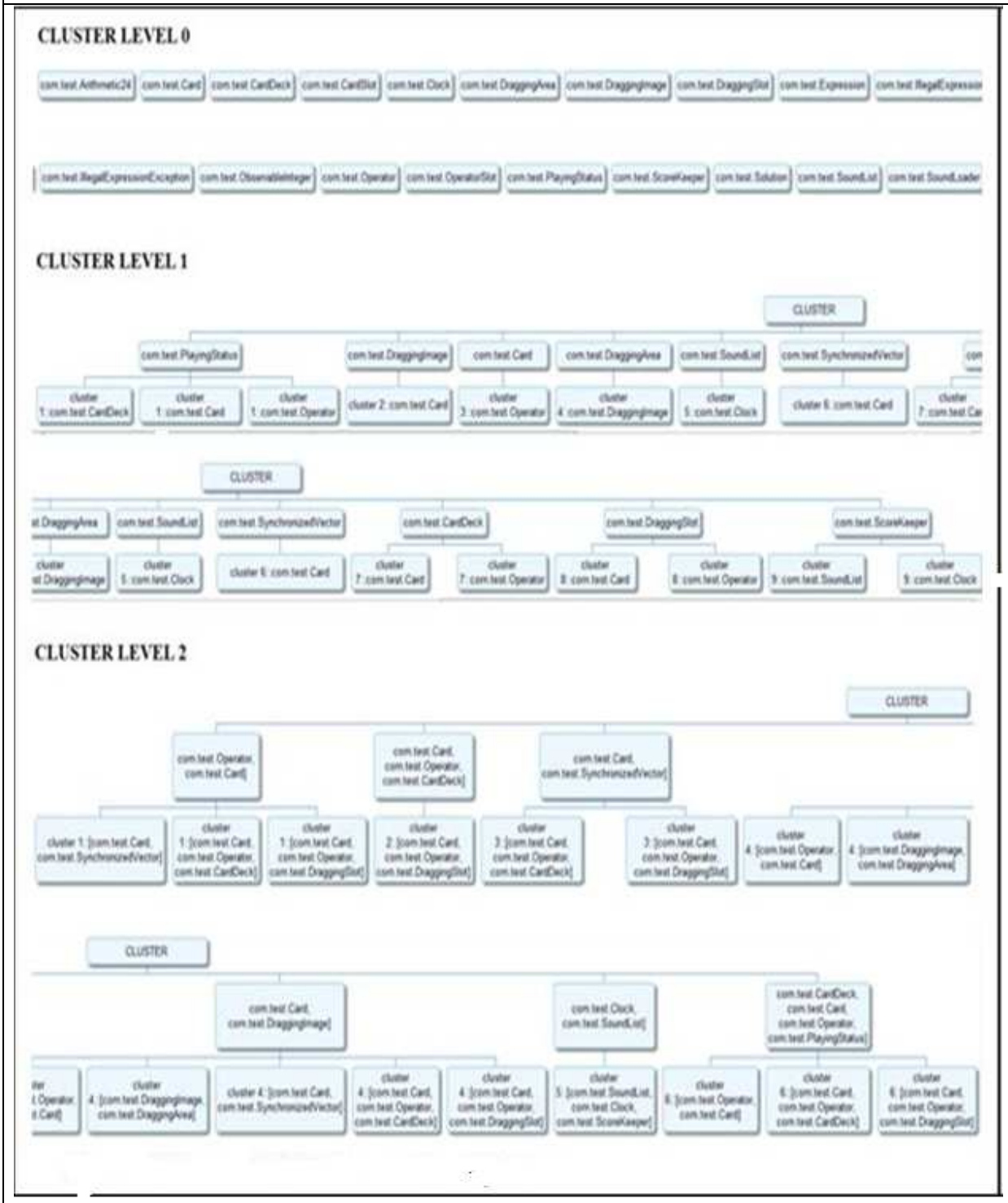


Figure 6.5: Remaining cluster levels created for 'Arithmetic24' game

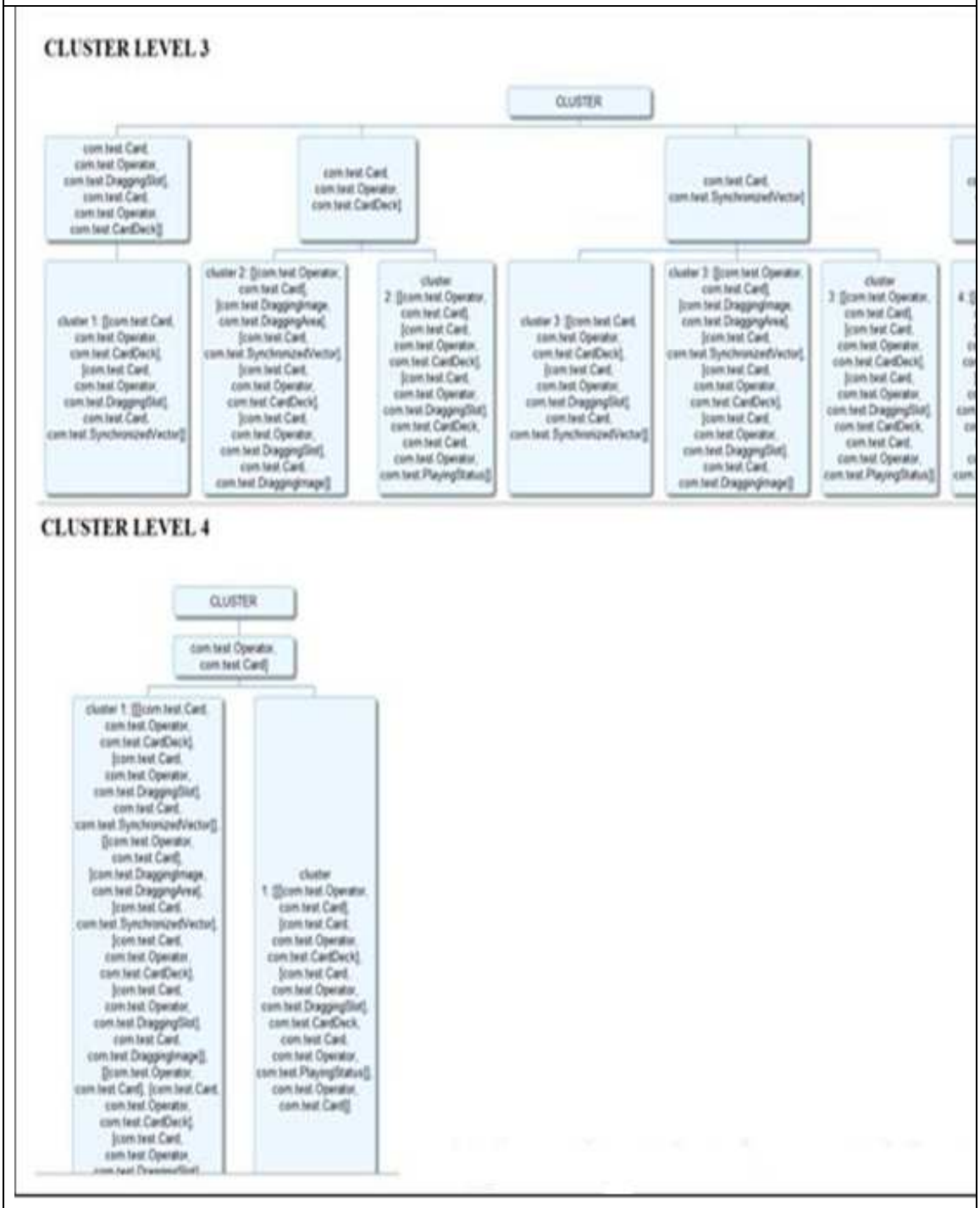
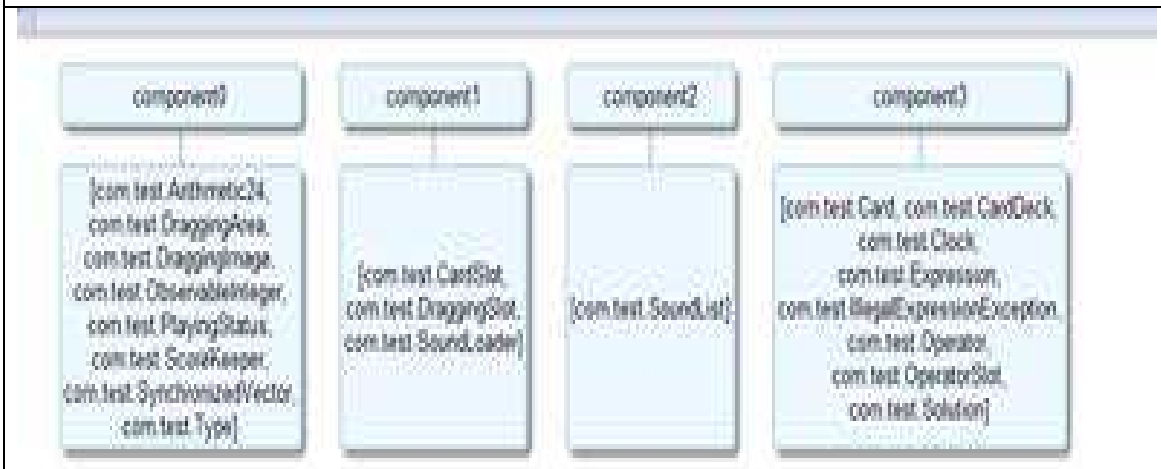


Figure 6.6: Components created for 'Arithmetic24' game



6.3 Module 3: Component evaluation and interface identification.

Objective:

To identify interface details among the components and to evaluate components quality using metrics like size, coupling and cohesion.

Strategy: Using components created in module – 2, component dependency is identified to show components are interacting with each other. These interfaces will work as provided and required interface for the component, which are nothing but the connectors. These interface details are used to access the quality of components using quality metrics.

Algorithms implemented: We have implemented the following algorithms –

Algorithm: 5.2.4 Component evaluation using size metric

Algorithm: 5.2.5 Component evaluation using Coupling metric

Algorithm: 5.2.6 interface details identification

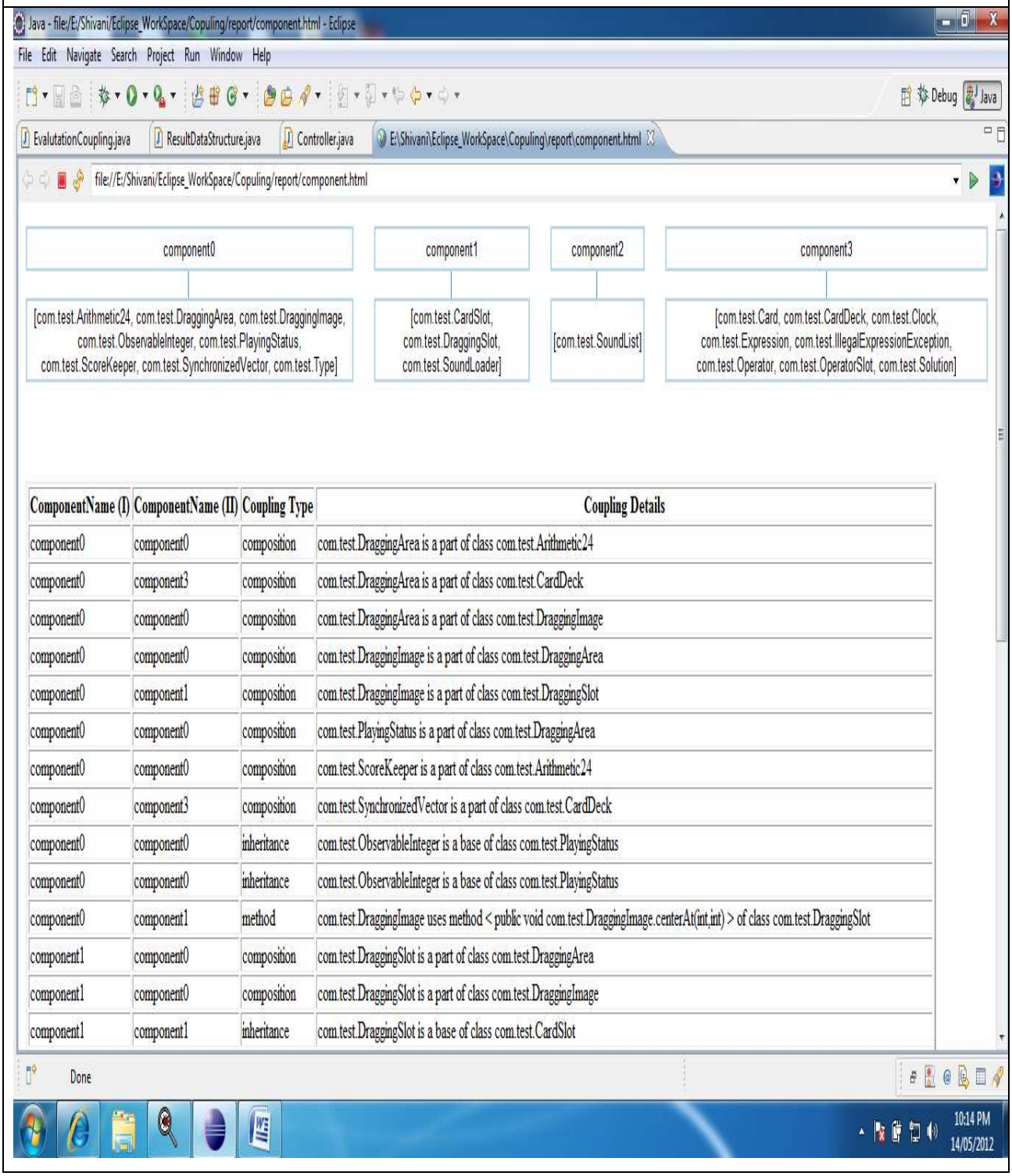
Algorithm: 5.2.7 Component evaluation using cohesion metric are implemented to get the results from module-3.

We have also written and executed some supporting programs to identify interface details and component evaluation.

Results from module 3

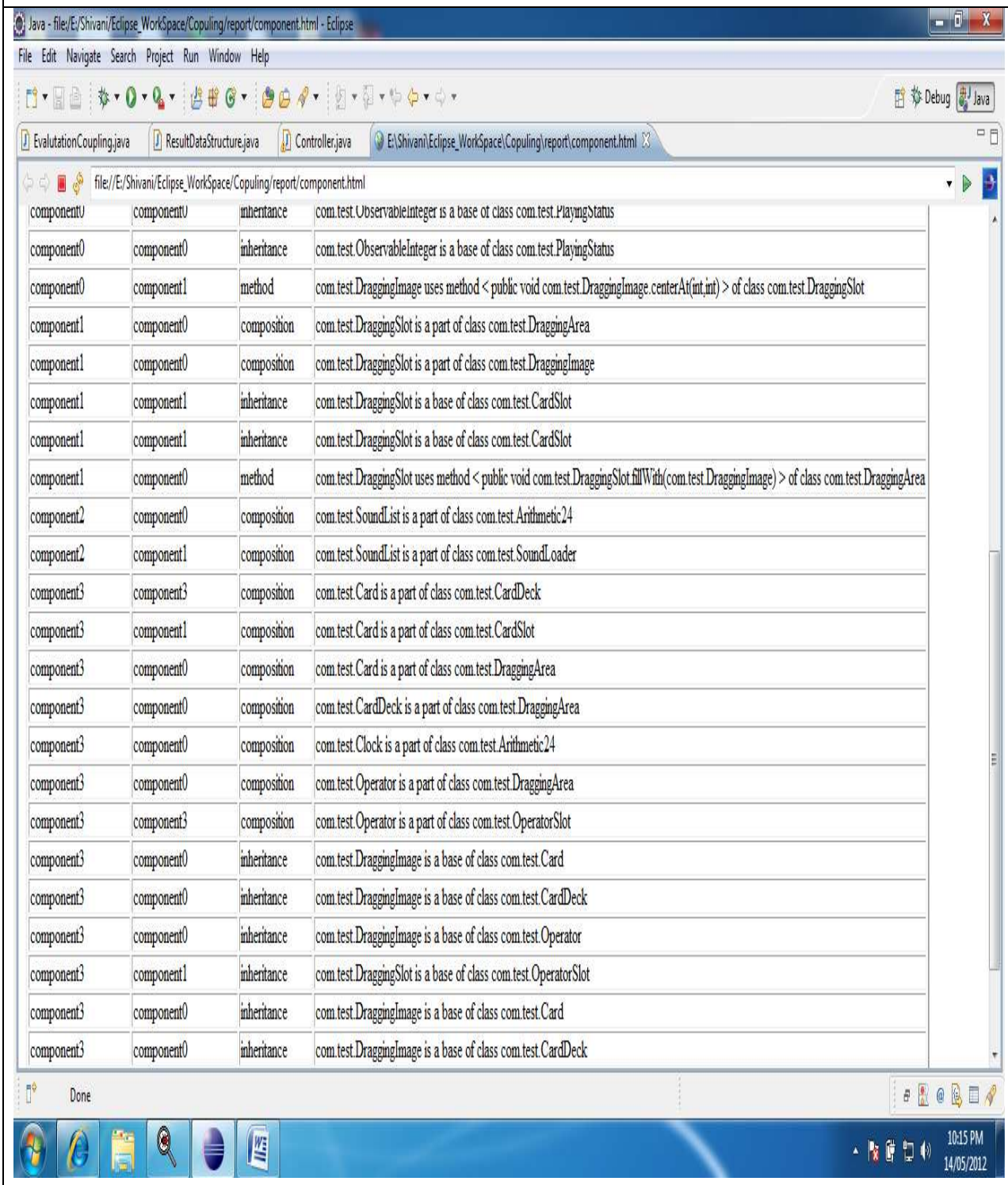
The figure.6.7 shows interface details created for these components. Using these details, interfaces among components can be created. “Table-6.1” show candidate components created along with respective classes for used case study “Arithmetic24 Game”. Using interface details component diagram with dependencies is shown in “fig.6.8 a”. Components are evaluated for quality using metrics size, coupling among components and cohesion within component and the results are shown in figure 6.7. The first evaluation metric chosen is the size evaluation criteria to show well organized components with appropriate number of implementation classes. So using size we evaluate clustering results. For this metric, sum of ratios of single class component, classes in largest component and other intermediate components should be 100%.i.e. sum of these should be 1.

Figure 6.7: Components created and interface details among components



ComponentName (I)	ComponentName (II)	Coupling Type	Coupling Details
component0	component0	composition	com.test.DraggingArea is a part of class com.test.Arithmetic24
component0	component3	composition	com.test.DraggingArea is a part of class com.test.CardDeck
component0	component0	composition	com.test.DraggingArea is a part of class com.test.DraggingImage
component0	component0	composition	com.test.DraggingImage is a part of class com.test.DraggingArea
component0	component1	composition	com.test.DraggingImage is a part of class com.test.DraggingSlot
component0	component0	composition	com.test.PlayingStatus is a part of class com.test.DraggingArea
component0	component0	composition	com.test.ScoreKeeper is a part of class com.test.Arithmetic24
component0	component3	composition	com.test.SynchronizedVector is a part of class com.test.CardDeck
component0	component0	inheritance	com.test.ObservableInteger is a base of class com.test.PlayingStatus
component0	component0	inheritance	com.test.ObservableInteger is a base of class com.test.PlayingStatus
component0	component1	method	com.test.DraggingImage uses method < public void com.test.DraggingImage.centerAt(int,int) > of class com.test.DraggingSlot
component1	component0	composition	com.test.DraggingSlot is a part of class com.test.DraggingArea
component1	component0	composition	com.test.DraggingSlot is a part of class com.test.DraggingImage
component1	component1	inheritance	com.test.DraggingSlot is a base of class com.test.CardSlot

Figure 6.7 continued……..



component0	component0	inheritance	com.test.ObservableInteger is a base of class com.test.PlayingStatus
component0	component0	inheritance	com.test.ObservableInteger is a base of class com.test.PlayingStatus
component0	component1	method	com.test.DraggingImage uses method < public void com.test.DraggingImage.centerAt(int,int) > of class com.test.DraggingSlot
component1	component0	composition	com.test.DraggingSlot is a part of class com.test.DraggingArea
component1	component0	composition	com.test.DraggingSlot is a part of class com.test.DraggingImage
component1	component1	inheritance	com.test.DraggingSlot is a base of class com.test.CardSlot
component1	component1	inheritance	com.test.DraggingSlot is a base of class com.test.CardSlot
component1	component0	method	com.test.DraggingSlot uses method < public void com.test.DraggingSlot.fillWith(com.test.DraggingImage) > of class com.test.DraggingArea
component2	component0	composition	com.test.SoundList is a part of class com.test.Arithmetic24
component2	component1	composition	com.test.SoundList is a part of class com.test.SoundLoader
component3	component3	composition	com.test.Card is a part of class com.test.CardDeck
component3	component1	composition	com.test.Card is a part of class com.test.CardSlot
component3	component0	composition	com.test.Card is a part of class com.test.DraggingArea
component3	component0	composition	com.test.CardDeck is a part of class com.test.DraggingArea
component3	component0	composition	com.test.Clock is a part of class com.test.Arithmetic24
component3	component0	composition	com.test.Operator is a part of class com.test.DraggingArea
component3	component3	composition	com.test.Operator is a part of class com.test.OperatorSlot
component3	component0	inheritance	com.test.DraggingImage is a base of class com.test.Card
component3	component0	inheritance	com.test.DraggingImage is a base of class com.test.CardDeck
component3	component0	inheritance	com.test.DraggingImage is a base of class com.test.Operator
component3	component1	inheritance	com.test.DraggingSlot is a base of class com.test.OperatorSlot
component3	component0	inheritance	com.test.DraggingImage is a base of class com.test.Card
component3	component0	inheritance	com.test.DraggingImage is a base of class com.test.CardDeck

Figure 6.7 continued……..

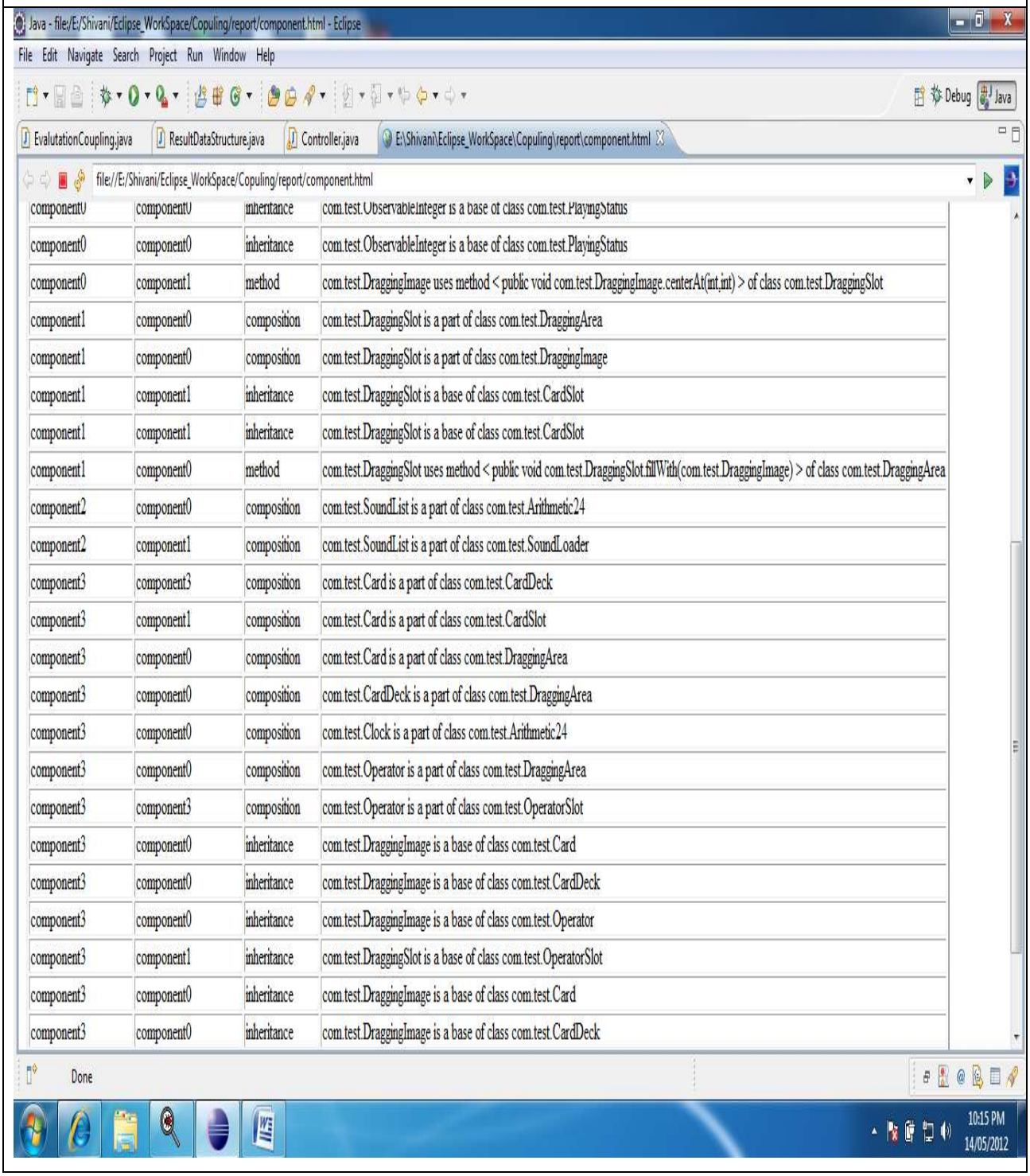


Figure 6.7 continued……..

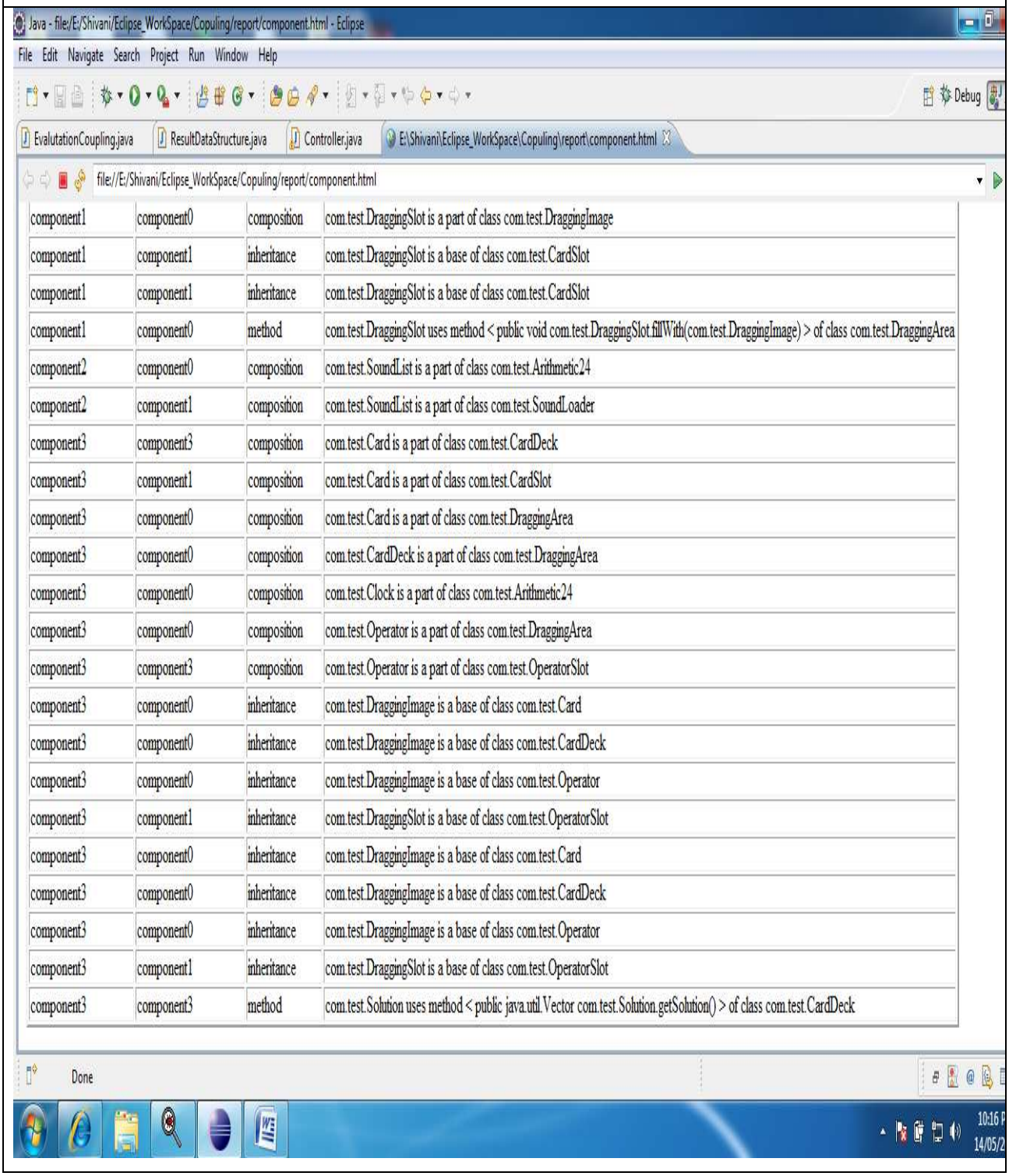
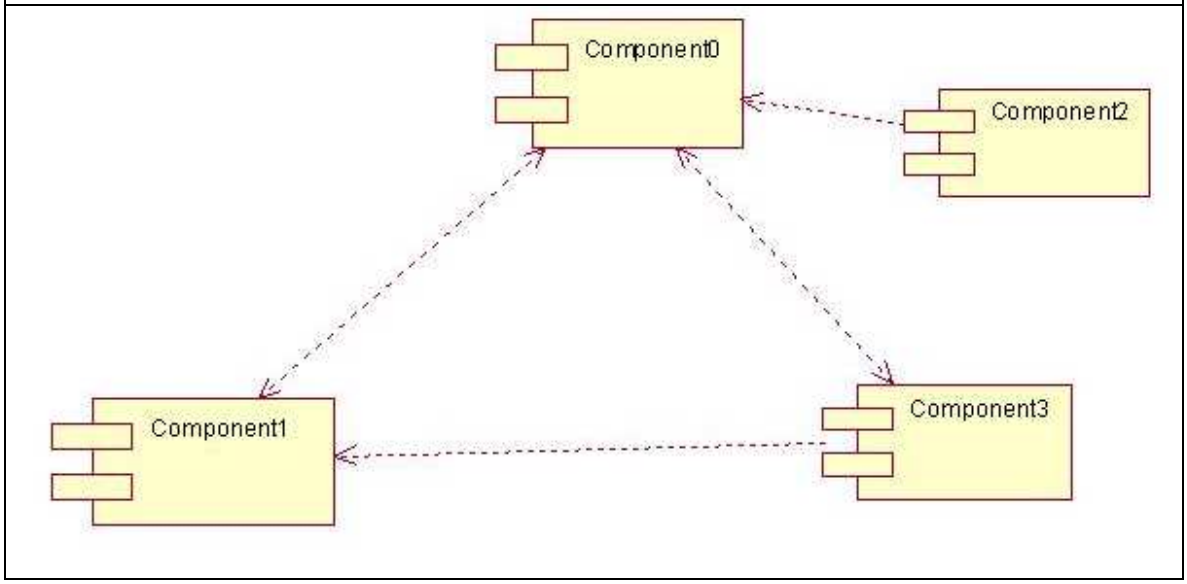


Figure 6.8 a) UML Component Diagram for Arithmetic24 game



Candidate components	Classes
Component0	Arithmetic24,DraggingArea,DraggingImage,ObservableInteger,PlayingStatus, ScoreKeeper,SynchronizedVector,Type
Component1	CardSlot, DraggingSlot,SoundLoader
Component2	SoundList
Component3	Card, CardDeck, Clock, Expression, IllegalExpressionException, Operator, OperatorSlot, Solution

Table 6.1: Candidate components recovered from Proposed approach & tool for "Arithmetic24" game

Figure 6.8 b) UML Components with interfaces as packages for Arithmetic24 game

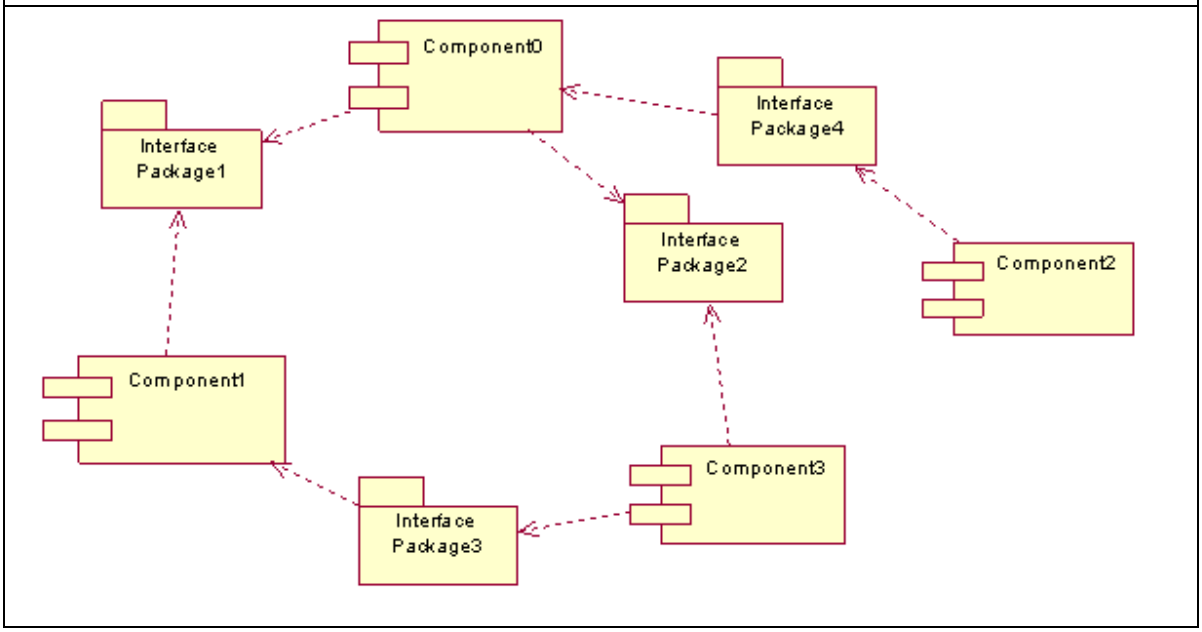
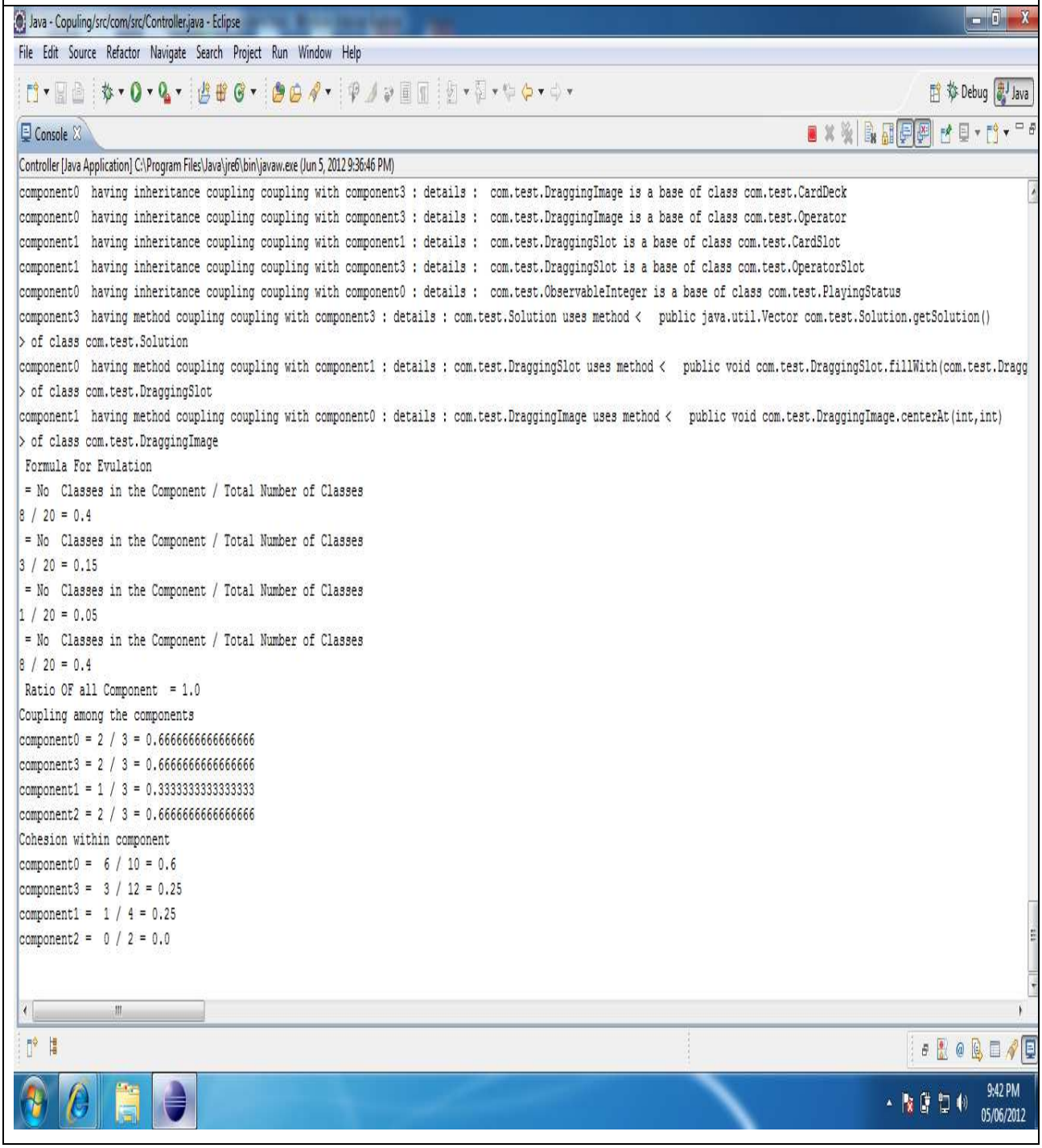


Figure 6.9: Component Evaluation by using Component Size, Component Coupling and Component Cohesion Metrics



Result Analysis

Figure 6.1 and figure 6.2 shows all the classes are extracted and different coupling tables are displayed. Result shows most of the classes are placed in proper coupling tables. We have identifies 20 classes(19 classes & 1 interface) from the given input. We have compared the result with the class diagram generated with the tool Enterprise Architecture. Initial experimental results from a case study were encouraging. The tool successfully extracted the classes and identifies coupling dependencies and displayed it in tabular format.

Using cluster levels, components are created for "Arithmetic24 Game". We have identified four components for the same, as shown in figure 6.6.We have also identified 20 classes of "Arithmetic24" game. We have compared the result with the class diagram generated with the tool Enterprise Architecture. Figure 6.6 shows these 20 classes are placed in four components by proposed approach.

So from figure 6.7 and "Table 6.1" largest components are component0 and component3 consisting of 8 classes each. So Ratio of classes in largest component0 =8/20 = 40% and Ratio of classes in largest component3 =8/20 = 40%. There is a single class component, component2, so Ratio of Single class component=1/20 = 5%.There is one intermediate component, component1, so Ratio of other intermediate components = 3/20 = 15%. Thus sum of these three ratios is 100%; it indicates all the classes in the software have been considered by three ratios. Also Result screen "fig.6.9" shows evaluation of components by coupling metric. Coupled component Ratio (CCR) for Component0 = 0.66, CCR for Component1=0.33, CCR for Component2=0.66, CCR for Component3=0.66.Again from result screen "fig.6.9" shows evaluation of components by Component Cohesion Metric (CCM). CCM for Component0=0.6, CCM for Component1=0.25, CCM for Component2=0, CCM for Component3=0.25. "Fig.6.8 a" shows dependencies among components created through proposed tool. Component dependencies must be decreased. We decrease dependency by managing interfaces into another package. So using interface

package, components with cyclic dependency can be removed, as shown in “Fig.6.8 b”. We can create component packages and interface packages which will play role of required interface and provided interface. Deployment of components and interfaces will depend upon the framework you use.

6.4 Sample Case studies – Analysis Chart

To assess the results from proposed study, we used six different small and medium size systems developed in java as input to the proposed tool. These experiments aimed to evaluate the tool for producing components of good quality. In this section we present experimental results and extracted artifacts from proposed tool for these six systems.

Table 6.2 summarizes the result from the proposed tool for various system.

System Name ⟶	Arithmetic24	Shopping Cart	Feedback Analysis	MyTool	CD project	Satellite
Number of classes& interface extracted	20	31	36	45	11	12
Number of Method coupling identified	03	00	0	08	04	02
Number of composition coupling identified	11	01	0	06	01	05
Number of Inheritance coupling identified	03	00	0	00	00	00
Number of integration coupling	12	01	0	14	04	06
Number of Cluster levels created	04	01	01	02	02	01
Number of component identified	04	01	01	01	01	01
Time	4.20 sec	6 min	7.15 sec	10 min	2.30 sec	2.48 sec

Table: 6.2 Sample Case studies – Analysis Chart

Figure 6.10 below shows various artifacts extracted by proposed tool for various systems.

All the classes and couplings like method coupling, composition coupling, inheritance coupling and integrated coupling of classes are properly extracted. We can also see appropriate cluster levels and components are created. Closely related classes are grouped together to form component. For shopping cart and feedback analysis system almost all the classes are unrelated. So the distance between any two classes is 1 or near to 1,hence just single component is created for these system. For the systems MyTool, CD project and satellite very few dependencies are there, hence it is obvious for cluster calculation distance goes to near 1 and again single component is created. In case of Arithmetic24 system all the three basic dependencies are present, hence integrated coupling is calculated properly and all 20 classes of the system are placed properly into four components. We are using integrated coupling, hence at each cluster level, distance calculation of class with every other class or clustered group is considered repetitively for similarity measure. Hence, if number of classes are more it takes more time to form components. We compare these results with our first phase of generating class diagram with existing reverse engineering tool Rational Rose and EA. We found that all the classes and dependencies have been extracted by proposed tool for all the systems tested. Hence components are generated properly. Performance of the tool is shown in figure 6.11.

Figure 6.10: Sample Case Studies Analysis Chart

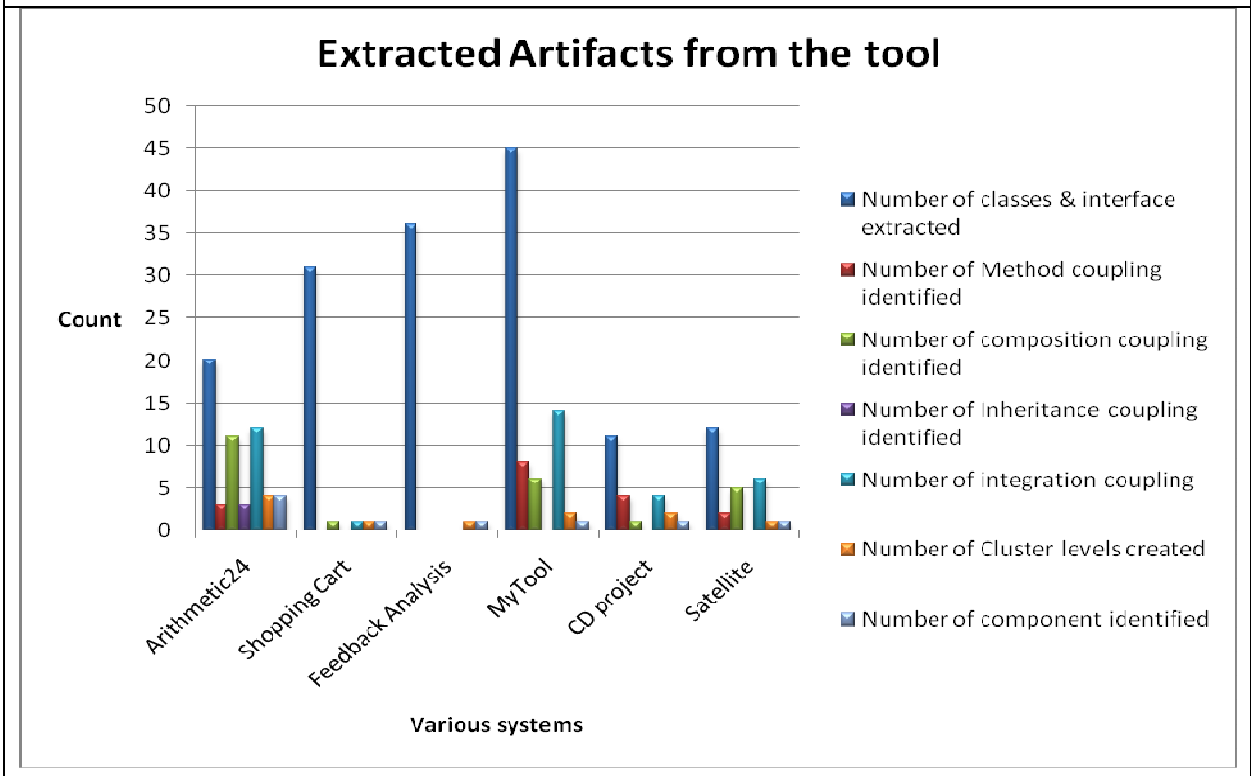
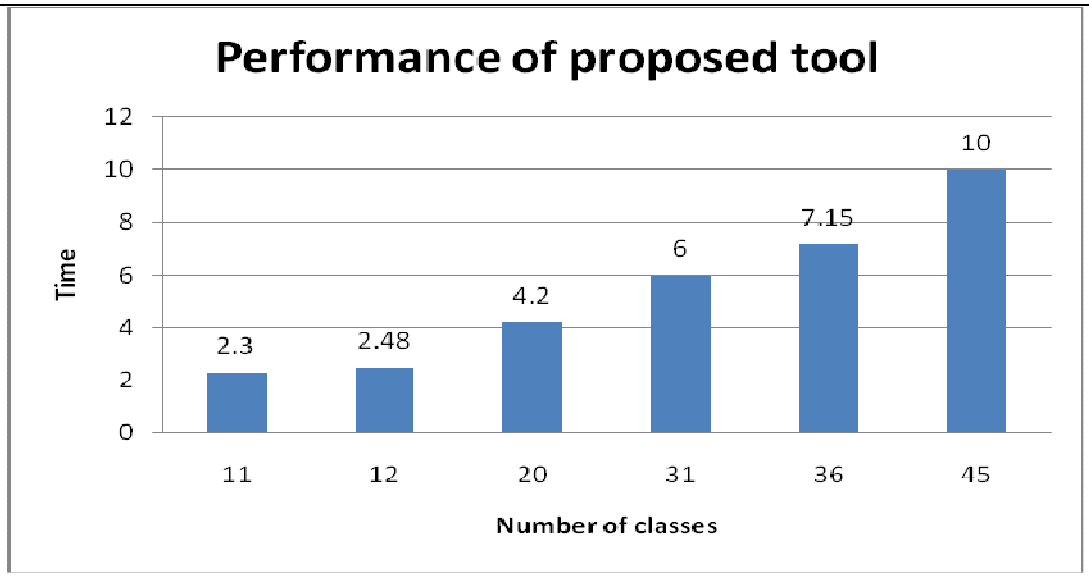


Figure 6.11: Performance of Proposed Tool



6.5 Comparative Study of Proposed Tool verses Existing Approaches

Following table 6.3 shows comparison of proposed approach with other approaches. We have considered manual approaches like FOCUS, CRUD method and object-z method. We have also considered automatic methods like ROMANTIC, Lee method and RCA method along with semiautomatic Hassan method. We have made comparison on the basis of extraction of elements required for component based architecture and evaluating components for quality using the most important metrics like cohesion, coupling and size of component. The methods are described in detail in chapter 2. From the table 6.3 we can see that all the methods are able to identify components. FOCUS method and Lee method partially identifies connector details. ROMANTIC ,Hassan method and RCA do not identify connectors but CRUD method, and object-z method identifies connectors. But CRUD and object-z are manual methods. ROMANTIC , Lee and Hassan method used clustering techniques for recovering artifacts. Lee method partially used cohesion and coupling component evaluation.RCA method supports cohesion and coupling. Component size metric is not used by any of these methods. The proposed tool supports all the details required for component based system also the approach is automatic. It can also be observed from table 6.3 that proposed tool gives all the circumstances required for component based system.

Approach	Identify components	Identify connectors	Clustering classes	Cohesion evaluation	Coupling evaluation	Component size evaluation	Auto-mation level
FOCUS approach	S	P	N	N	N	N	manual
CRUD method	S	S	N	N	N	N	manual
ROMANTIC method	S	N	S	N	N	N	auto-matic
Lee method	S	P	S	P	P	N	auto-matic
Object-Z method	S	S	N	N	N	N	manual
Hassan method	S	N	S	N	N	N	Semi-auto
RCA method	S	N	N	S	S	N	auto-matic
Proposed tool	S	S	S	S	S	S	auto-matic

Table 6.3: Comparison of the proposed tool and other approaches

Here in the table 6.3,

S – Supports

P- Partially support

N – Not used

6.6 Research outcome

From the above experiments' results and analysis we can infer that –

- The research study conducted will help providing assistance to software maintenance for transforming existing object oriented system to component based system.
- Through research study reuse of existing code and migrating to new environment becomes easy and it saves cost, efforts of redesign and redeveloping the system which suits to new evolving environment. This is what the software industry always prefers.
- The powerful tool will assist to extract components and interface details from object oriented system to form component based system.
- The research study gives maximum automation and less human intervention ,that will reduce human efforts and cost of software development.
- The tool itself evaluate extracted components for quality.
- It also help management by reducing human efforts and cost saving.

The work in this chapter summarizes the results got from small and medium size software application.

Chapter 7

Summary and Conclusion

7.1 Summary

Software architecture gives high level of abstraction of system and plays very important role in at least six aspects of software development: understanding, reuse, construction, evolution, analysis and management. However, the original architecture of software would deviate from actual system, due to software maintenance and software evolution. Most of the times software architecture documents are not available.

Today, computing environments are evolving from mainframe systems to distributed system. Standalone programs developed using object oriented technologies are not suitable for these new computing environments. Instead programs developed using component based technology has proven to be more suitable for new environments due to their granularity and reusability. For this reason components can be used more effectively and are better suited for reuse than the objects from a object oriented system. We can get maintainability and reliability of software by reusing existing elements and classes in legacy object oriented system. Therefore, we should derive reusable components from classes in object oriented system and change the object oriented system into component based system. Components within system interact with each other through required-interfaces and provided-interfaces. These interfaces act as connectors between components. This gave a direction to pursue research in the area of component based software architecture recovery from object oriented system.

Software architecture recovery methods are classified according to process input used, approach and techniques used to extract architecture. Whichever is the input used or approach used; manual techniques and semi- automatic techniques are time consuming and require lot of human efforts. The research focused on quasi-automatic method and implemented it into automatic tool. The proposed approach and tool extracted components and interface details from object oriented system. Clustering has been applied for gaining architectural understanding and recovering component based architecture of object oriented software systems. Relationship among the classes played a very important role during clustering, as they are used to determine similarity between entities to be clustered.

Our research study is mainly divided into two phases:

- Extract the classes of given source code using existing tool available
- Develop tool for component identification and interface details generation.

First phase is a kind of analysis phase of a legacy object oriented system through existing reverse engineering tool. The main objective here was to examine different existing reverse engineering tools, access the capabilities of tools and choose best of them to generate static structure of object oriented system. We have chosen here four reverse engineering tools; commercial and non-commercial. This is required to generate UML class diagram, which shows different classes and static relationship among the classes. The research assumes that no documentation is available of legacy system. Hence, this class diagram is useful to verify results from first module of proposed framework. Here we observed that most of the classes are extracted by all the four tools (Rational Rose, Enterprise Architecture, Reverse and Argo UML) but all the relationships have not been extracted properly. We concluded that Rational rose and Enterprise Architecture extracts maximum required static information. Hence, any one available tool can be used to generate class diagram.

Second phase of the research is proposing framework, tool and implementation of it, which is again divided into three modules. Objective of this phase was to automate the

approach of component creation, interface identification and component evaluation. The three modules are listed below.

- Identify dependencies in existing object oriented system

  The main objective here was to find inheritance coupling, composition coupling and method coupling and integrated coupling of these three couplings. We considered these important coupling dependencies as they are basis for identifying components from object oriented system. Components are required to create meaningful connectors.

- Identify components

  The objective here was to group the similar classes together to form the components using existing dependencies among classes and propose, implement agglomerative hierarchical clustering algorithm.

- Component evaluation and interface identification.

  The objective here was to identify interface details among the components and to evaluate components quality using metrics like size, coupling and cohesion.

To evaluate the proposed approach and tool for producing components of good quality, we used six different small and medium size systems developed in java as input to our tool. The proposed tool identified existing dependencies among the classes of these object oriented systems. The result showed that all the classes from object oriented system are extracted and various dependencies are displayed by tool. We have compared the result with the class diagram generated with the tool Enterprise Architecture. Relationship among the classes are used to determine similarity between entities to be clustered, hence module 1 is important and forms a basis for module 2. In module 2, we created components based on relationships extracted using agglomerative clustering algorithm. Results showed that appropriate cluster levels are created and based on that components are created. These components are evaluated for quality by using cohesion, coupling and size metric of component, in the module 3.Interface details of the identified

components are also displayed in the result. We have studied and provide guidelines for implementing these components and interfaces in OSGi framework, which is one of the popular frameworks for implementing component based system. We have written various algorithms and implemented them in our tool. We have used different java applications to evaluate our tool. Results of the tool are satisfactory and showed reduced time and human efforts than other methods available. Results also showed various artifacts extracted from various object oriented system and performance of our tool. We developed tool in java to migrate java applications into component based system.

Summary of Results Obtained:

Primary objective of this research was to come up with a tool that will help software maintenance person to migrate object oriented systems into a component based system.

This research will

- help software maintenance to migrate existing object oriented system to component based system.
- reuse existing code while migrating to new environment
- save cost, efforts of redesign and redeveloping the system which suits to new evolving environment.
- assist in extracting components and interface details from object oriented system to form component based system.
- reduce human intervention by maximum automation.
- evaluate the extracted components for quality.
- help management in cost saving.

7.2 Conclusions

As stated earlier, components have more granularity and reusability than the classes and are suitable for new distributed computing environment. Software industry is migrating to component based technology. Component identification is a critical part of software

reengineering. In this research study, we have proposed and implemented Framework, tool to recover component based architecture from an object oriented system. The research study conducted on “Extraction of connector classes from object oriented system while recovering software architecture” helped in migrating legacy object oriented system into component based system. The research study comprises of three modules. Before starting the first module, we need to identify static structure of object oriented system. This can be done by retrieving class diagram of the object oriented system. For retrieving class diagram, we have examined four existing reverse engineering tools - IBM Rational Rose, Enterprise Architecture, Reverse and ArgoUML. Post this, we have compared results gained from these tools and selected a tool which retrieves maximum static information. Since we are recovering software architecture of a system whose design documentation is not available, the static information retrieved from reverse engineering tool is used to compare the results from module-1 of our tool.

Module-1 constitutes identifying existing dependencies in the object oriented system. Existing relationships in the object oriented code helps to group the related classes together in the form of components hence this module is designed and implemented. Here we have considered important relationships in any object oriented systems i.e. inheritance relationship, composition relationship and method calls from one class to other classes in the system.

Module-2 constitutes of identifying components from object oriented system. We proposed distance calculation function to find similarity between classes of object oriented system. We have implemented similarity distance calculation algorithm and agglomerative clustering algorithm to group similar classes into one component. We have used 6 small and medium size object oriented applications developed in java to test, how our tool creates components based on the existing relationships in the object oriented application.

Module-3 constitutes identifying interface details of component identified on module-2.These interface details are used to create connectors of components i.e. required and provided interface of components. These required and provided interfaces help to

components to communicate with each other. Components created in Module-2 are evaluated here for quality using quality metrics component coupling, component cohesion and component size. The proposed approach shows these evaluations also.

Thus, these three modules are implemented in the tool by using java language. We have studied and provided guidelines for deploying these components created and interface details into OSGi framework. We have conducted experiments on six small and medium size object oriented systems. Experimental results showed that our tool gave satisfactory result in terms of clustering quality. It was effective for software architecture recovery. Even though hierarchical clustering is time consuming, it is better than manual and semi-automatic approaches which require much more time than hierarchical clustering.

7.3 Suggestions for Further Research

There are various avenues for further research in this research study. More specifically, some of the areas which can be further investigated are listed below:

- The hierarchical clustering method is highly time consuming process, especially when it is employed in large-scale software system. Improving the efficiency of agglomerative hierarchical clustering algorithm will be considered in future research.
- Identification of components and interfaces, by considering dynamic relationship and dynamic interaction of classes, would be one of the future works.
- Identified components and connector storage and retrieval could also be considered as future work.

References

[1] Aline.P.V. Vasconcelos, and C.M.L. Werner, "Software Architecture Recovery based on Dynamic Analysis", XVIII Brazilian Symposium on Software Engineering, Workshop on Modern Software Maintenance, 2, Brasilia, DF, Brazil, October, 2004.

[2] Abdelkrim Amirat and Mourad Oussalah , “Enhanced Connectors to Support Hierarchical Dependencies in Software Architecture”, ACM NOTERE 2008, June23-27, Lyon, France. pp. 257-266, 2008

[3] Alae-Eddine El Hamdouni, A.Djamel Seriai1, and Marianne Huchard, ” Component based architecture recovery from OO systems via relational Concept Analysis”, CLA10 7th International Conference on Concept Lattices and Their Applications, Sevilla : Spain (2010) pp. 259-270 ,2010

[4] Ali Shokoufandeh, Spiros Mancoridis, Trip Denton, Matthew Maycock, ” Spectral and meta-heuristic algorithms for software clustering”, The Journal of Systems and Software, vol 77,No.3 pp. 213-223,2004

[5] Andrey A.Terekhov, ” Dealing with Architectural Issues: a Case Study”, ACM SIGSOFT Software Engineering Notes Volume 29 Number 2,pp. 1-4,2004

[6] Andre L. C. Tavares, Marco Tulio Valente,” A Gentle Introduction to OSGi”, ACM SIGSOFT Software Engineering Notes, Volume 33 Number 5 pp 1-5, 2008

[7] Arie van Deursen, Christine Hofmeister, Rainer Koschke, Leon Moonen,and Claudio Riva. Symphony, “View-driven software architecture reconstruction”, In WICSA, pp. 22–134, 2004.

[8] Bridget Spitznagel and David Garlan, “A Compositional Approach for Constructing Connectors”, Proceedings of the Working IEEE/IFIP Conference on Software Architecture ,WICSA 2001

[9] Brian S. Mitchell, Spiros Mancoridis and Martin Traverso , ” Search Based Reverse Engineering” SEKE '02, Ischia, Italy pp. 431-438 July 15-19, 2002

[10] Brian S. Mitchell and Spiros Mancoridis , “On the Automatic Modularization of Software Systems Using the Bunch Tool”, IEEE Trcactionsans On Software Engineering, VOL. 32, NO. 3, pp. 193-208 MARCH 2006

[11] Brian S. Mitchell and Spiros Mancoridis, “On the evaluation of the bunch search-based software modularization algorithm”, Soft Comput., 12(1) pp. 77–93, 2008.

[12] Chandrashekar Rajaraman, Michael R. Lyu, “Some Coupling Measures for C++ Programs”.

[13] Chung-Horng Lung, Marzia Zaman, Amit Nandi, “Applications of Clustering Techniques to Software Partitioning, Recovery and Restructuring”, Journal of Systems and Software - Special issue: Applications of statistics in software engineering, Volume 73 Issue 2, pp. 227 - 244 Elsevier Science Inc. New York, NY, USA October 2004

[14] Chung-Horng Lung, “Software Architecture Recovery and Restructuring through Clustering Techniques”, Proceedings of the 3rd International Software 1 Architecture Workshop (ISAW), pp.101-104, 1998

[15] Coad P., Yourdan E. (1991) “Object oriented Design,”. Prince-hall, Englewood cliffs, NJ.

[16] D.H.Hutchens and V.R. Basili, ”System structure Analysis: clustering with Data Bindings,”. IEEE transactions software Engineering Vol 11 No 8, pp 749-757, August 1985

[17] Danny Lange and Yuichi Nakamura “Interactive visualization of design patterns can help in framework understanding”. In OOPSLA, pp. 342–357, 1995.

[18] Eunjoo Lee Byungjeong Lee Woochang Shin Chisu Wu , “A Reengineering Process for Migrating from an Object-oriented Legacy System to a Component-based System”, Proceedings of the 27th Annual International Computer Software and Applications Conference , COMPSAC’03,2003

[19] Frank Simon ,Silvio Loffler, Claus Lewerentz. Al, “Distance based cohesion measuring”. Accepted for FESMA99, Amsterdam 4. –8. October ,1999.

[20] G. Murphy, D. Notkin, and K. Sullivan. “Software reflexion models:Bridging the gap between source and high-level models”. In SIGSOFT, pp. 18–28. ACM Press, 1995.

[21] Gabriela Ar´evalo, St´ephane Ducasse and Oscar Nierstrasz, “Lessons Learned in Applying Formal Concept Analysis to Reverse Engineering”, ICFCA'05 Proceedings of the Third international conference on Formal Concept Analysis, pp. 95-112 ,2005.

[22] Gabriela Ar´evalo and Tom Mens, “Analysing Object Oriented Framework Reuse using Concept Analysis”, In ECOOP 2002: Proceedings of the Inheritance Workshop, A.Black, E.Ernst, P.Grogono and M. Sakkinen (Eds.), pp. 3-9, 2002.

[23] 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.

[24] Ganter, B., Wille, R., “Formal Concept Analysis — Mathematical Foundations”, Springer, 1999.

[25] Garlan, “Software Architecture : a roadmap”, In ICSE- Future of SE track, pp. 91-101,2000.

[26] George Yanbing Guo, Atlee, and Kazman, “A software architecture reconstruction method” In WICSA, pp. 15–34, 1999.

[27] Ghulam Rasool, and Nadim Asif ,“ Software Architecture Recovery”, International Journal of Computer, Information, and Systems Science, and Engineering 1;3 pp. 206-211 , 2007

[28] Hassan Gomaa, Daniel A. Menascé and Michael E. Shin, “Reusable Component Interconnection Patterns for Distributed Software Architectures”, SSR’01, May 18-20, Toronto, Ontario, Canada pp. 69-77, 2001.

[29] Hassan Mathkour, Ameur Touir, Hind Hakami, Ghazy Assassa , “On the transformation of object oriented-based systems to Component based Systems”, IEEE International Conference on Signal Image Technology and Internet Based Systems ’08, pp. 11-15, 2008.

[30] Hausi A. Müller, Kenny Wong, and Scott R. Tilley, “Understanding software systems using reverse engineering technology”, In OO Tech. for Database and Soft. Sys., pp.240–252. World Scientific, 1995.

[31] Helgo M. Ohlenbusch and George T. Heineman, “Composition and interfaces within software architecture”, Proceedings of the 1998 conference of the Centre for Advanced Studies on Collaborative research, CASCON’98.pp 17, 1998.

[32] Hemant Jain, Naresh Chalimeda, Navin Ivaturi ,Balarama Reddy, “Business Component Identification- A Formal Approach”, Proceedings of the Fifth International Enterprise Distributed Object Computing Conference (EDOC’01) pp.183-187,2001.

[33] Hironori Washizaki and Yoshiaki Fukazawa, “A technique for automatic component extraction from object-oriented programs by refactoring”. Sci. Comput. Program., 56(1-2) pp. 99–116, 2005

[34] Houari A. Sahraoui, Hakim Lounis, Walcelio Melo, and Hafedh Mili, “A concept formation based approach to object identification in procedural code”, In Automated Software Engineering Journal, Volume 6 No 4, Kluwer Academic Publishers, 1999, pp. 387-410.

[35] IBM Rational Rose Enterprise Edition software.

[36] Igor Ivkovic and Michael W. Godfred, “Architecture recovery of Dynamically Linked Applications: A case study”, Proceedings of the 10 th International Workshop on Program Comprehension (IWPC’02),2002

[37] Ivkovic and Godfrey, “Enhancing domain-specific software architecture recovery”, In the proceedings of IWPC, pp 266–276, 2003

[38] Istvan Gergely Czibula and Gabriela¸Serban,“ Hierarchical Clustering for Software Systems Restructuring”, 2007

[39] Ivan T. Bowman and Richard C. Holt,“ Software architecture recovery using Conway's law”, in the proceedings of the 1998 conference of the Centre for Advanced Studies on Collaborative Research CASCON '98 ,pp 1-11,1998.

[40] Jain A.M,Murthy M.N and Flynn P.J. Dam,“ Clustering: A review”, ACM Computing surveys,31(3); pp. 264-323,1999.

[41] James Sasitorn and Robert Cartwright, “Deriving Components from Genericity”, SAC’07 March 11-15, Seoul, Korea, ACM, pp. 1109- 1116, 2007.

[42] Jian Feng Cui,Heung Seok Chae, “Applying agglomerative hierarchical clustering algorithms to component identification for legacy systems”, Information and Software technology, 53, pp. 601-614 , 2011.

[43] Jiawei Han and Micheline Kamber, “Data Mining Concepts and Techniques”, second edition, Elsevier publisher,2006.

[44] Jonas Lundberg and Welf L¨owe, “Architecture Recovery by Semi-Automatic Component Identification”, Electronic Notes in Theoretical Computer Science 82 No. 5, Published by Elsevier Science B. V., 2003.

[45] Jong Kook Lee, Seung Jae Jung, Soo Dong Kim, Woo Hyun Jang, Dong Han Ham, “Component Identification method with coupling and cohesion” In the proceedings of the Eight Asia Pacific Software Engineering Conference,APSEC’01, IEEE, 2001.

[46] Kamran Sartipi and Kostas Kontogiannis,“ Component Clustering Based on Maximal Association” Proceedings of the Eighth Working Conference On Reverse Engineering , pp. 1-12 (WCRE.01),2001.

[47] Lee E., B. Lee ,W. Shin and C.Wu, “A reengineering process for Migrating from an object oriented Legacy system to Component based system”, In proceedings of the 27th International Computer Software and Aplication Conference(COMPSAC),Dallas,TX, USA, Nov 3-6 IEEE Computer Science press, pp.336-341,2003.

[48] Lei Ding and, Nenad Medvidovic “Focus: A Light-Weight, Incremental Approach to Software Architecture Recovery and Evolution”, Proceedings of the Working IEEE/IFIP Conference on Software Architecture , WICSA '01 pp. 191-200, 2001.

[49] Maher Salah and Spiros Mancoridis, “A hierarchy of dynamic software views: from object-interactions to feature-interacions”. In ICSM, pp. 72–81, IEEE Press, 2004.

[50] Marie Chavent, “A monothetic clustering method”, Pattern Recognition Letters, Volume 19, Issue 11, pp. 989−996 September 1998.

[51] Mircea Lungu and Michele Lanza, Tudor Gˆırba,,“ Package Patterns for Visual Architecture Recovery”, In Proceedings of European Conference on Software Maintenance and Reengineering (CSMR 06),2006.

[52] Nadim Asif, “Architecture Recovery”, In the Proc. of International Conference of Information and Knowledge Engineering (IKE’02), Las Vegas, 2002.

[53] Naouel Moha, Amine Mohamed Rouane Hacene, Petko Valtchev, andYann-Ga¨el Gu´eh´eneuc, “Refactorings of Design Defects using Relational Concept Analysis”, ICFCA'08 Proceedings of the 6th international conference on Formal concept analysis, pp. 289-304,2008.

[54] Nenad Medvidovic , Alexander Egyed and Paul Gruenbacher, “Stemming Architectural Erosion by Coupling Architectural Discovery and Recovery”, Proceedings of 2nd Second International Workshop from Software Requirements to Architectures(STRAW), collocated with ICSE 2003, Portland,Oregon,May2003.

[55] Nenad Medvidovic and Vladimir Jakobac, “Using Software Evolution to Focus Architectural Recovery”, Journal Automated Software Engineering, volume 13, Issue 2, pp. 225-256 April 2006 .

[56] Nicolas Anquetil and Timothy C. Lethbridge, “Recovering software architecture from the names of source files”, Journal of Software Maintenance, 11 pp. 201–221, 1999.

[57] O’Brien, L., “Dali: A Software Architecture Reconstruction Workbench”, Software Engineering Institute, Carnegie Mellon University, May 2001.

[58] Oleksandr Grygorash, Yan Zhou, Zach Jorgensen, “Minimum Spanning Tree Based Clustering Algorithms”, ICTAI '06 Proceedings of the 18th IEEE International Conference on Tools with Artificial Intelligence, pp. 73-81, 2006.

[59] O. Maqbool, H.A. Babri,“ The Weighted Combined Algorithm: A Linkage Algorithm for Software Clustering”, Proceedings of the Eighth European Conference on Software Maintenance and Reengineering, CSMR’04, IEEE, 2004.

[60] Onaiza Maqbool and Haroon A. Babri, "Hierarchical Clustering for Software Architecture Recovery", IEEE Transactions On Software Engineering, Vol. 33, No. 11, pp. 759 - 780 November 2007.

[61] Ondrej Galik and Tomas Bures"Generating Connectors for Heterogeneous Deployment", Proceedings of the 5th International workshop on Software Engineering and middleware (SEM '05) September Lisbon, Portugal pp.54-61,2005.

[62] Pang- Ning Tan, Michael Steinbach, Vipin Kumar," Introduction to Data Mining", Published by Pearson Education Inc.,2006.

[63] Pascal Andr´e, Nicolas Anquetil, Gilles Ardourel, Jean-Claude Royer," Component types and communication channels recovery from Java source code", WCRE, Lille: France, 2009.

[64] Pollet, D., Ducasse, S., Poyet, L., Alloui, I., Cimpan, S., Verjus, H.," Towards a process oriented software architecture reconstruction taxonomy", In: CSMR '07: Proceedings of the 11th European Conference on Software Maintenance and Reengineering. pp. 137–148. IEEE Computer Society, Washington, DC, USA, 2007.

[65] Prasanta K. Jana and Azad Naik," An Efficient Minimum Spanning Tree based Clustering Algorithm", Methods and models in computer science, ICM2CS ,pp. .1-5, 2009.

[66] Qifeng Zhang, Dehong, Qiu, Qubo Tian, Lei Sun," Object Oriented Software Architecture Recovery using New Hybrid Clustering Algorithm", Seventh Internationl conference on Fuzzy systems and Knowledge Discovery (FSKD 2010), 2010.

[67] Rainer Koschke, Daniel Simon," Hierarchical Reexion Models", Proceedings of 10th working conference on Reverse Enginnering, pp 36-45, 2003.

[68] Rick Kazman and S. Jeromy Carriere," View extraction and view fusion in architectural understanding", In International Conference on Software Reuse, 1998.

[69] Rick Kazman and S. J. Carriere, "Playing detective: Reconstructing software architecture from available evidence", Automated Soft. Engineer., 1999.

[70] Rick Kazman, Liam O'Brien, and Chris Verhoef," Architecture reconstruction guidelines. Technical report", Carnegie Mellon Univ., SEI, 2001.

[71] Robert Allen and David Garlan,"Formalizing Architectural Connection", 0270-5257/94 IEEE, pp. 71- 80, 1994.

[72] Robert Allen and David Garlan," A Formal Basis for Architectural Connection" Journal ACM transactions on Software Engineering and Methodology, TOSEM , volume 6 issue 3, pp. 213-249 July 1997.

[73] Roger S. Pressman, "Software Engineering A practitioner's Approach", Sixth Edition, McGraw Hill Publications.

[74] S.K.Mishra, Dr.D.S.Kushwaha, and Prof.A.K.Misra," Creating Reusable Software Component from Object-Oriented Legacy System through Reverse Engineering", in Journal of Object Technology, pp. 133-152 Jan-Feb 2009.

[75] S. Mancoridis, B. S. Mitchell , Y. Chen, E. R. Gansner, Bunch: A Clustering Tool for the Recovery and Maintenance of Software System Structures", Proceedings of IEEE International Conference on Software maintenance, pp. 50-59,1999.

[76] Shaheda Akthar, Sk.Md.Rafi,**"** Improving The Software Architecture Through Fuzzy Clustering Technique", Indian Journal of Computer Science and Engineering ISSN : 0976-5166 Vol 1 No 1 pp. 54 – 57,2010.

[77] Shaheda Akthar and Sk.MD.Rafi, "Recovery of Software Architecture Using Partitioning Approach by Fiedler Vector and Clustering**",** Computer and Information Science Vol.3, No.1, pp. 72-75, February 2010.

[78] Shivani Budhkar and Arpita Gopal," Reverse Engineering Java Code to Class Diagram: An Experience Report", in International Journal of Computer Applications (0975 – 8887) Volume 29– No.6, September 2011, pp. 36-43.

[79] Shivani Budhkar and Arpita Gopal," Component based software architecture recovery from object oriented system using existing dependencies among classes", ,in International Journal of Computational Intelligence Techniques ISSN: 0976-0466 & E-ISSN: 0976-0474, Volume 3, Issue 1, 2012, pp.-56-59.

[80] Component identification from existing object oriented system using Hierarchical clustering," in IOSR Journal of Engineering, ISSN: 2250-3021, May. 2012, Vol. 2(5) pp: 1064-1068.

[81] Siff, M., Reps, T.W, "Identifying modules via concept analysis" IEEE Trans. Software Eng. 25(6), pp. 749–768 ,1999.

[82] Siraj Muhammad, Onaiza Maqbool, Abdul Qudus Abbas ," Role of relationship during clustering of object oriented software system", 6th International conference on Emerging technologies (ICET), 2010.

[83] Simon Allier, Salah Sadou, Houari Sahraoui and Regis Fleurquin "From Object Oriented Applications to Component Oriented Application via Component Oriented Architecture", Ninth working IEEE/IFIP conference on Software Architecture, 2011.

[84] Simon Allier , Houari A. Sahraoui and Salah Sadou, "Identifying Components in Object-Oriented Programs using Dynamic Analysis and Clustering", in proceedings of the 2009 conference of the Centre for Advanced studied on Collaborative research,CASCON'09', pp. 136-148.

[85] Smeda, A., Oussalah, M., and Khammaci, T, "Improving Component-Based Software Architecture by Separating Computations from Interactions" In Proceedings of the ECOOP Workshop on Coordination and Adaptation Techniques for Software Entities (WCAT '04), Oslo, Norway, 2004.

[86] Soo Ho Chang, Man Jib Han, and Soo Dong Kim, "A Tool to Automate Component Clustering and Identification", M. Cerioli (Ed.): FASE 2005, LNCS 3442, pp. 141 – 144,Springer-Verlag Berlin Heidelberg, 2005.

[87] Spiros Mancoridis and Brian S. Mitchell, "Using automatic clustering to produce high-level system organizations of source codes", In IWPC. IEEE Press, 1998.

[88] Spiros Xanthos, "Identification of Reusable Components within an Object- oriented Software System using Algebraic Graph Theory", OOPSLA'04, October 24-28,Vancouver, British Columbia , Canada, pp. 322-323, 2004.

[89] Stéphane Ducasse, Tudor Gîrba, Michele Lanza, and Serge Demeyer, "Moose: a collaborative and extensible reengineering environment", In Tools for Software Maintenance and Reengineering, RCOST/Software Technology, pp. 55–71, 2005.

[90] Stephen Kell, "Rethinking Software Connectors", SYANCO 07 September pp.- 1-12 , Dubrovnik, Croatia, 2007

[91] Suk Kyung Shin and Soo Dong Kim, "A Method to transform Object oriented Design into Component based Design using Object-Z", Proceedings of Third international Conference on Software Engineering Research, Management and Applications,SERA'05,IEEE,2005.

[92] Sylvain Chardigny, Abdelhak Seriai, Dalila Tamzalit, Mourad Oussalah, "Quality-Driven Extraction of a Component-based Architecture from an Object-Oriented System", in the proceedings of the 2008 12th European Conference on Software maintenance and Reengineering, CSMR '08 pp. 269-273,2008.

[93] Sylvain Chardigny, Abdelhak Seriai, Mourad Oussalah, Dalila Tamzalit, "Extraction of Component-Based Architecture From Object-Oriented Systems", Seventh Working IEEE/IFIP Conference on Software Architecture , pp. 285 – 288, 2008.

[94] T.A. Wiggerts, "Using clustering algorithms in Legacy system Remodularization", In the proceedings of the 4th working Conference on Reverse Engineering, WCRE'97, IEEE,1997.

[95] Thomas Tilley, Richard Cole, Peter Becker, and Peter Eklund, A survey of formal concept analysis support for software engineering activities"'In ICFCA. Springer-Verlag, 2003.

[96] Trevor Parsons, Adrian Mos, Mircea Trofin, Thomas Gschwind, " Extracting Interactions in Component-Based Systems", IEEE Transactions on Software Engineering, vol. 34, No. 6, pp. 783- 799 November/December 2008.

[97] Tzerpos V. and Holt R. C, A Hybrid Process for Recovering Software Architecture.", In the proceedings of CASCON'96, Toronto, 1996.

[98] Van Deursen, A., Kuipers, T, "Identifying objects using cluster and concept analysis", In: ICSE. pp. 246–255, 1999.

[99] Vijayan Sugumaran, Veda C. Storey" A Semantic-Based Approach to Component Retrieval", The DATA BASE for Advances in Information Systems - Summer 2003 Vol. 34, No. 3,pp. 8-24,2003.

[100] Wolfgang Eixelsberger, Lasse Warholm , Rene Klösch , Harald Gall and Berndt Bellay," A Framework for Software Architecture Recovery" Proceedings of International Conference on Software Engineering (ICSE '97), Boston, USA, May 1997.

[101] Wolfgang Eixelsberger, Michaela Ogris, Harald Gall, Berndt Bellay, " Software Architecture Recovery of a Program Family", pp. 508-511 IEEE, 1998.

[102] Woo-Jin Lee, Oh-Cheon Kwon, Min-Jung Kim, and Gyu-Sang Shin, " A Method and Tool for Identifying Domain Components Using Object Usage Information", ETRI Journal, Volume 25, Number 2, pp.- 121-132 ,April 2003.

[103] Xiaojin Zhu, "Clustering", CS769 Spring 2010 Advanced Natural Language Processing, 2010.

[104] Xinyu Wang,Xiaohu Yang,Jianling Sun and Zhengong Cai, "A New Approach of Component Identification Based on Weighted Connectivity Stength Metrics", Information Technology Journal 7(1), pp. 56-62,2008.

[105] Yanbing Guo, Atlee, and Kazman, "A software architecture reconstruction method". In WICSA, pp. 15–34, 1999.

[106] Yuxin Wang, Ping Liu, He Guo , han Li, Xin Chen, " Improved Hierarchical Clustering algorithm for Software Architecture Recovery", International Conference on Intelligent Computing and Cognitive Informatics, 2010.

[107] Young Ran Yu, Soo Dong Kim ,Dong Kwan Kim, "Connector Modeling Method for Component Extraction"IEEE, pp. 46-53,1999.

[108] Zhongjie Wang, Xiaofei Xu, and Dechen Zhan, " A Survey of Business Component Identification Methods and Related Techniques" International Journal of Information Technology Volume 2 Number 4, pp 229-238, 2005.

Reference Sites

[109] ArgoUML, http://argouml.tigris.org/ is a widely used open source tool for UML modeling tool

[110] http://www.neiljohan.com/projects/reverse/ :- Reverse is non commercial tool to convert java code to class diagram developed by Neil Johan.

[111] http://www.sparxsystems.com/products/ea/downloads.html:-Enterprise Architecture (EA) is widely used commercial UML modeling tool

[112] http://en.wikipedia.org/wiki/Software_architecture_recovery

Appendix – I

Glossary of relevant terms

- **Software Architecture:** Software architectures are composed of components, connectors and configurations, constraints on the arrangement and behavior of components and connectors. The architecture of a software system is a model, or abstraction of that system.
- **Software architecture recovery: Software architecture recovery** is a set of methods for the extraction of architectural information from lower level representations of a software system, such as source code. The abstraction process to generate architectural elements frequently involves clustering source code entities (such as files, classes, functions etc.) into subsystems according to a set of criteria that can be application dependent or not.
- **Class:** In object-oriented programming, a class is a template definition of the method s and variable s in a particular kind of object.
- **Object:** An object is a specific instance of a class; it contains real values instead of variables.
- **Cluster:** A group of the same or similar elements gathered or occurring closely together.
- **Clustering:** Clustering is the process of forming groups of items or such that entities within a group are similar to one another and different from those in other groups.

- **Agglomerative:** It is type of hierarchical clustering.This is a "bottom up" approach: each observation starts in its own cluster, and pairs of clusters are merged as one moves up the hierarchy.
- **Component:** a component is an identifiable part of a larger program or construction. Usually, a component provides a particular function or group of related functions. In programming design, a system is divided into components that in turn are made up of modules. In short, a component is group of classes collaborating to provide a function of application.
- **Connectors:** Connectors represents interaction among components. From the run time perspective, connectors mediate the communication and coordination activities among components.
- **Interface:** An interface defines the signature operations of an entity; it also sets the communication boundary between two entities, in this case two pieces of software. It generally refers to an abstraction that an asset provides of itself to the outside. The main idea of an interface is to separate functions from implementations. Any request that matches the signature or interface of an object may also be sent to that object, regardless of its implementation. The concept of an interface is fundamental in most object oriented programming languages.
- **Reverse Engineering:** It is the part of software engineering, which consists process of recreating design by analyzing a final product.
- **Reverse Engineer:** A software engineer, who is responsible for performing reverse engineering.
- **Software maintenance:** The software maintenance is modification of a software product after delivery to correct faults, to improve performance or other attributes, or to adapt the product to a modified environment.

- **Similarity:** Similarity measures determine how similar a pair of entities is, in clustering process.
- **Dissimilarity:** The dissimilarity between two objects is a numerical measure of the degree to which the two objects are different. The common interval for dissimilarity is [0, 1] but can range from 0 to ∞.

Appendix – II (a)

Experimental Environment

All programs were written in Java (version 1.6) Language under Eclipse Galileo version. The experiments were conducted on a 2-GHz Intel (R) Pentium(R) P6100 CPU with 4 GB bytes of RAM running Windows 7 ultimate version.

Sample Source code:

MatrixController.java

```
package com.src;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;
import com.datastructure.Properties;
import com.datastructure.UnionIntersectionMatrix;
import com.util.Intersection;
import com.util.Union;
public class MatrixController
{
        private Properties properties;
```

```
	private ArrayList<UnionIntersectionMatrix> unionIntersectionMatrixs;
	public MatrixController()
	{
		// TODO Auto-generated constructor stub
		unionIntersectionMatrixs = new
ArrayList<UnionIntersectionMatrix>();
	}
	public Properties getProperties() {
		return properties;
	}
public void setProperties(Properties properties) {
	this.properties = properties;
	}
	public ArrayList<UnionIntersectionMatrix> getUnionIntersectionMatrixs() {
	return unionIntersectionMatrixs;
	}
public void setUnionIntersectionMatrixs(ArrayList<UnionIntersectionMatrix>
unionIntersectionMatrixs) {
		this.unionIntersectionMatrixs = unionIntersectionMatrixs;
	}
	public MatrixController(Properties properties) {
		this();
		this.properties = properties;
```

```
                String [] classNameArray = properties.getKey();
                for (int i = 0; i < classNameArray.length; i++)
                {
                        for (int j = i + 1; j < classNameArray.length; j++)
                        {
        ArrayList<String> sourceList = properties.getValue(classNameArray[i]);
        ArrayList<String> destList = properties.getValue(classNameArray[j]);
        int unionCount = Union.calclualteUnion(sourceList, destList);
int intersectionCount = Intersection.calculateIntersection(sourceList, destList)
        this.unionIntersectionMatrixs.add(new
UnionIntersectionMatrix(classNameArray[i],classNameArray[j],unionCount,intersect
ionCount));
 }
}
} }
```

Cluster .java

```
package com.src;
import java.util.ArrayList;
import java.util.Iterator;
import com.datastructure.Properties;
import com.datastructure.UnionIntersectionMatrix;
import com.util.CompositePropertyMatrix;
public class Cluster
```

```
{
        private Properties compositeProperties;
        private ArrayList<UnionIntersectionMatrix> listUnionInterMatrix;
public
Cluster(CompositePropertyMatrixcompositePropertyMatrix,ArrayList<UnionIntersec
tionMatrix> listUnionInterMatrix)
        {
                super();
                if(compositePropertyMatrix != null  && listUnionInterMatrix != null)
                {
                this.compositeProperties = compositePropertyMatrix.getProperties();
                        this.listUnionInterMatrix = listUnionInterMatrix;
                }
        }
public Cluster(Properties compositeProperties,ArrayList<UnionIntersectionMatrix>
listUnionInterMatrix)
        {
                super();
                if(compositeProperties != null  && listUnionInterMatrix != null)
                {
                        this.compositeProperties = compositeProperties;
                        this.listUnionInterMatrix = listUnionInterMatrix;
                }
        }
```

```
public Properties createCluster()
{
        Properties clusterProperties  = null;
        try
        {
if(this.compositeProperties != null  && this.listUnionInterMatrix != null)
                {
                        clusterProperties = new Properties();
                        // create a one single cluster for each classes..
                String [] classArray = this.compositeProperties.getKey();
                        for (int i = 0; i < classArray.length; i++)
                        {
                                // starting this list will be empty
                        ArrayList<String> list = new ArrayList<String>();
                for (int j = i + 1; j < this.listUnionInterMatrix.size(); j++)
                                {
UnionIntersectionMatrix intersectionMatrix = this.listUnionInterMatrix.get(j);
if( intersectionMatrix.getSourceClassName().equalsIgnoreCase(classArray[i]) )
                                        {
                                                // to get the distance
                                                // to get the destination class
                if(intersectionMatrix.distance() <= 0.70) {
```

```
        list.add(intersectionMatrix.getDestClassName());
                                                    }
                                                }
                                            }
                clusterProperties.put( "Cluster [" + classArray[i] + "]", list );
                                    }
                                }
                }
                catch(Exception exception)
                {
                        exception.printStackTrace();
                }
                return  clusterProperties;
        }
}
```

Appendix – II (b)

Clustering Concepts

The clustering techniques can be used effectively to facilitate software Architecture recovery. Clustering is identified as “quasi-automatic” technique for software architecture reconstruction and recovery. We present here the clustering concepts used for software Architecture Recovery followed by method adopted in this research, study of existing reverse engineering tool and study of component based framework.

Overview of Clustering

Unsupervised classification or clustering is considered as most important unsupervised learning problem. Clustering techniques have been used in many disciplines to support grouping of similar objects of a system. This is one of the most fundamental techniques adopted in science and engineering. The ability to form meaningful groups of objects is one of the most fundamental modes of intelligence. Clustering is the process of grouping objects into clusters such that the objects from the same clusters are similar and objects from different clusters are dissimilar. Objects can be described in terms of measurements (for example, attributes, features) or by relationships with other objects (for example pair wise distance, similarity).The inputs required for clustering process are similarity measures or data from which similarities can be computed. The primary objective of clustering analysis is to facilitate better understanding of the observations and the subsequent construction of complex knowledge structure from features and object clusters. The key concept of clustering is to group similar things into clusters, such that intra-cluster similarity or cohesion is high, and inter-cluster similar or coupling is low. Coupling has great impact on many quality attributes, such as maintainability, verifiability, flexibility, portability, reusability, interoperability, and expandability. Thus, the main objective of clustering is similar to that of software partitioning described by Chung-Horng Lung [13].

Cluster analysis is used in a number of applications such as data analysis, image processing, market analysis, software architecture etc. Clustering helps in gaining, overall distribution of patterns and correlation among data objects.

Jiawei Han [42] defined clustering as it is data mining activity for differentiating groups (classes or clusters) inside given set of objects so that objects within clusters have high similarity in comparison to one another but are very dissimilar to objects in other clusters.

Onaiza Maqbool and Haroon A. Babri [59] defined; Clustering is the process of forming groups of items or such that entities within a group are similar to one another and different from those in other groups. The similarity between entities is determined based on their characteristics or features.

Many clustering algorithms have been presented in the literature, but they comprise of the following three common key steps:

- Obtain the data set.
- Compute the resemblance coefficients for the data set.
- Execute the clustering method.

According to Chung-Horng Lung [13], an input data set is an object-attribute data matrix. Objects are the entities that we want to group based on their similarities. Attributes are the properties of the objects. A resemblance coefficient for a given pair of objects shows the degree of similarity or dissimilarity between these two objects, depending on the way the data represents.

Categories of Clustering

Most clustering algorithms for software architecture recovery are based on two popular techniques known as partitional and hierarchical clustering.

Partitional Clustering: Pang- Ning Tan [61] described, it is simply a division of the set of data objects into non-overlapping subsets (clusters) such that each data object is in exactly one subset. Partitional algorithms usually start with an initial partition consisting of certain number of clusters. The partition is then modified at every step such that some criterion is optimized while keeping the number of clusters constant. Sub categories of partitional algorithms include graph-theoretic mixture, mixture resolving and mode-seeking algorithms. Partitional algorithms require the number of clusters to be known in advance, which is difficult, if we do not have prior knowledge about the data set. Onaiza Maqbool described [59]; algorithms are computationally expensive because we seek to partition n items into c clusters, which, even for moderate values of n and c, may result in a very large number of partitions to choose from. According to Brian S [9] and Ali Shokoufandeh [4], to reduce the computational complexity of partitional algorithms, researchers have used heuristic-based approaches to facilitate software Architecture recovery. If number of clusters can be reasonably determined in advance, partitional algorithms can be used for producing clusters representing software systems. D.H.Hutchens [15] described, partitional algorithms produce flat decompositions, whereas the natural decomposition of a software system is usually presented as a nested decomposition or hierarchy. Rainer Koschke and Daniel Simon [66] described, these decompositions of modules into sub modules are especially useful for understanding large systems. Following are the categories of partitional algorithms.

- Squared Error Algorithms -The most intuitive and frequently used criterion function in partitional clustering techniques is the squared error criterion, which tends to work well with isolated and compact clusters. The k-means is the simplest and most commonly used algorithm employing a squared error criterion. It starts with a random initial partition and keeps reassigning the patterns to clusters based on the similarity between the pattern and the cluster centers until a convergence criterion is met (e.g., there is no reassignment of any pattern from one cluster to another, or the squared error ceases to decrease significantly after some number of iterations).The k-means algorithm is popular because it is easy to implement, and its time complexity is O(n), where n is the number of patterns. Several variants of the k-means algorithm have been reported in the literature. Some of them attempt to select a good initial partition so that the algorithm is more likely to find the global minimum value.

Simple k-means Algorithm

Xiaojin Zhu, [102] presented a widely used clustering algorithm. It assumes that we know the number of clusters k. This is an iterative algorithm which keeps track of the cluster centers (means). The centers are in the same feature space as x.

1. Randomly choose k centers μ1, . . . , μk.

2. Repeat

3. Assign x1 . . . xn to their nearest centers, respectively.

4. Update μi to the mean of the items assigned to it.

5. Until the clusters no longer change.

Step 3 is equivalent to creating a Voronoi diagram under the current centers. K-means clustering is sensitive to the initial cluster centers. It is in fact an optimization problem with a lot of local optima1. It is of course sensitive to k too. Both should be chosen with care.

-Graph-Theoretic Clustering -The best-known graph-theoretic divisive clustering algorithm is based on construction of the minimal spanning tree (MST) of the data, and then deleting the MST edges with the largest lengths to generate clusters. The hierarchical approaches are also related to graph-theoretic clustering. Single-link clusters are sub graphs of the minimum spanning tree of the data which are also the connected components. Complete-link clusters are maximal complete sub graphs, and are related to the node color ability of graphs.

T.A. Wiggerts [93] described, graph theoretic algorithms work on graphs. The nodes of such graphs represent entities and the edges represent relations. Graph algorithms do not start from the individual nodes (entities), but try to find sub graphs which will form the clusters. Special kinds of sub graphs like connected components, maximal complete sub graphs or spanning trees are used to derive modules or are candidates themselves. The algorithms used to find these special sub graphs are provided by or based on graph theory. Often applied examples of algorithms which fit in this category are Minimal Spanning Tree (MST) clustering and aggregate algorithms.

Minimal Spanning Tree: Oleksandr Grygorash [57] described, the MST clustering algorithm is known to be capable of detecting clusters with irregular boundaries. Once the MST is built for a given input, there are two different ways to produce a group of clusters. If the number of clusters k is given in advance, the simplest way to obtain k clusters is to sort the edges of the MST in descending order of their weights, and remove the edges with the first k − 1 heaviest weights. This approach is called the standard EMST (Euclidean Minimal Spanning Tree) clustering algorithm or SEMST (Standard Euclidean Minimal Spanning Tree). The second approach does not require a preset cluster number. Edges, that satisfy a predefined inconsistency measure, are removed from the tree. It is the inconsistency measure suggested by Zahn, and therefore it is called the clustering algorithm Zahn's EMST clustering algorithm or ZEMST.

The basic MST based clustering algorithm is as follows.

First construct MST using Kruskal algorithm and then set a threshold value and step size. We then remove those edges from the MST, whose lengths are greater than the threshold value. Then calculate the ratio between the intra-cluster distance and inter-cluster distance and record the ratio as well as the threshold. We update the threshold value by incrementing the step size. Every time we obtain the new (updated) threshold value, we repeat the above procedure. We stop repeating, when we encounter a situation such that the threshold value is maximum and as such no MST edges can be removed. In such a situation, all the data points belong to a single cluster. Finally, we obtain the minimum value of the recorded ratio and form the clusters corresponding to the stored threshold value. The above algorithm has two extreme cases:

1) With the zero threshold value, each point remains within a single cluster.
2) With the maximum threshold value all the points lie within a single cluster.

Therefore, the proposed algorithm searches for that optimum value of the threshold for which the Intra-Inter distance ratio is minimum. It need not be mentioned that this optimum value of the threshold must lie between these two extreme values of the threshold. However, in order to reduce the number of iterations, we never set the initial threshold value to zero.MST gives comparatively better performance than k-means algorithm. The disadvantage is Threshold value and step size needs to be defined apriori, described by Prasanta K. Jana and Azad Naik [64].

Aggregation Algorithms- Aggregation Algorithms reduce the number of nodes (representing entities) in a graph by merging them into aggregate nodes. The aggregate nodes can be used as cluster or can be the input for new iteration resulting in higher level aggregates.

T.A. Wiggerts [93] described; the graph reduction technique selects nodes (one at a time) and makes a new aggregate node containing the selected node together with its neighborhood set (the set of nodes no further than r edges away). For each node r is determined so that the resulting aggregate node will contain R nodes. So, R is the degree of reduction. Less supervised variants allow variable values for R.

Hierarchical Clustering: It permits clusters to have sub clusters. Hierarchical algorithms produce nested decomposition or hierarchy. When purpose of clustering is architecture recovery, a multiple level architecture view is important and facilitates architectural understanding. Hierarchical algorithm provides a view with earlier iterations presenting a detailed view of the architecture and later iterations presenting a high level view. Moreover, hierarchical algorithms do not require the number of clusters to be known in advance. The similarity between the entities is determined based on their characteristic or features.

Features: It represents characteristics of entities, on the basis of which their similarity is determined during clustering. The efficiency of clustering depends on careful selection of features. For software architecture recovery, researchers have mostly utilized static information of formal and non-formal features. Formal features includes functions called by an entity, global variables, macros and user defined types referred to by an entity, files included in an entity and classes in case of object oriented system. The non-formal features include comments, identifiers, developer names, directory path, LOC, time of last update.

General Hierarchical Clustering Process:

- Identify features and entities in the system and represent each entity as feature vector.
- Select similarity measure and develop n×n similarity matrix representing the similarity between every pair of entities within system.

- Selection of clustering algorithm to form clusters such that entities within a cluster are more similar to each other than to entities in other clusters, until the required number of clusters is formed or only one cluster remains.
- Selection of evaluation method: Evaluation of clusters for quality assessment can be performed by using internal assessment or external assessment. Internal assessment refers to an intrinsic evaluation of clustering results like cohesion, coupling of modules within decomposition. External assessment can be performed by comparing clustering results with manual recovery by experts.

There are two types of hierarchical clustering algorithms: agglomerative (bottom-up) and divisive (top-down).Both build a hierarchy of clustering in such a way that each level contains the same clusters as the first lower level except for two clusters which are joined to form one cluster. A hierarchical clustering is often displayed graphically using a tree-like diagram called a dendrogram, which displays both the cluster-sub cluster relationship and the order in which the clusters were merged(agglomerative view) or split (divisive view).

According to Jiawei Han [42] and Jain A. M. [39], for agglomerative hierarchical clustering, given a set of n objects, this algorithm begins with n singletons i.e. sets with one element, merging them until a single cluster is reached. The agglomerative clustering algorithms differ in the way two most similar clusters are calculated and the linkage metric used. The linkage metric are single linkage, complete linkage, average linkage. The single link algorithms merge the clusters whose distance between their closest objects is the smallest. Complete linkage algorithms merge the clusters whose distance between their most distant objects is the smallest. Average link algorithms merge the clusters whose average distance i.e. the average of distances between the objects from the clusters is smallest. One advantage of these algorithms is they are non- supervised. They do not need any extra information such as number of expected clusters and candidate regions of search space for locating each cluster.

General Agglomerative Method:

T.A. Wiggerts, [93] presented agglomerative hierarchical methods fit the following scheme, known as Johnson’s algorithm.

- Begin with N clusters each containing one entity, where N is number of entities and compute the similarities between the entities (clusters).
- While there is more than 1 cluster
- Do

    - Find the most similar pair of clusters
    - Merge these clusters into a single cluster
    - Update the similarities between the clusters

- End do

Often algorithms are presented in terms of dissimilarity. In this case the two clusters which are least dissimilar are joined. The different algorithms all follow the scheme above, however they use different parameters like similarity measures and updating rule. Updating rules are nothing but different linkage methods used like single linkage, complete linkage etc.

The Divisive (top-down) Methods start from one cluster containing all n objects and split it until n clusters are obtained. In each step a cluster is split into two clusters. After N-1 steps there are N clusters each containing one entity, N is the number of entities. Feasible divisive hierarchical methods can be either monothetic or polythetic.

Monothetic Methods: According to Marie Chavent [49], monothetic divisive clustering methods have first been proposed in the particular case of binary data. Since then, monothetic clustering methods have mostly been developed in the field of unsupervised learning and are known as descendant conceptual clustering methods.

These methods are mostly used with binary features. The division of clusters is determined by certain features (usually one) on which certain scores are necessary to belong to a certain new cluster. The best known variants of monothetic devise clustering are association analysis. In this method only one feature is used for the splitting. This result in cluster in which all entities possess that feature and a cluster in which no entity possess it. The splitting feature is chosen in such a way that the similarity between newly formed clusters is minimal in terms of a certain criterion. (e.g. information loss which should be maximized because it is dissimilarity measure.) In the next step of algorithm, another feature is selected for the splitting of the clusters. This need not be the same feature for all clusters. By following this procedure, the resulting hierarchy is equivalent to a decision tree in which each node is labeled with the feature used for splitting.

In **Polythetic Methods** the possession of a certain subset of the features suffices for an entity to belong to a cluster, no features are compulsory. Other definitions say that in polythetic methods all features are taken into account (e.g. to compute a similarity measure) where as monothetic methods only look at one feature at every level.

T.A. Wiggerts, [93] described, dissimilarity analysis, which is one of the most feasible polythetic methods. In this method a cluster A is split by taking out the entity a for which sim(a, A-{a}) is minimal (the original description by was in terms of dissimilarity). For this computation, several similarity measures working on an entity and a cluster can be used. Also the average Euclidean distance is used. The entity ‘a’ is used to form a new cluster, called splinter group. Now a number of iterations are performed. In each iteration, that entity which is the ‘more similar’ to S than to A is moved to S and the similarities are recomputed. The resulting clusters A and S are subdivided in the same way in the next step of the hierarchical algorithm.

Divisive algorithms offer an advantage over agglomerative clustering algorithms because most users are interested in the main structure of data which consists of few large clusters found in the first steps of divisive algorithms. Agglomerative algorithms start with the details (the individual entities) and work their way up to large clusters which may be affected by unfortunate decisions in the first steps. However Agglomerative hierarchical algorithms are most widely used for software architecture recovery. This is because it is infeasible to consider all possible divisions of the first large clusters (2N-1 - 1 possibilities in the first step).

Both the partitional and hierarchical clustering has been applied to facilitate software architecture recovery. Here in our study we focus on hierarchical clustering technique.

Different Types of Clusters

Clustering aims to find useful groups of objects (clusters), where usefulness is defined by the goals of the data analysis. Following are different types of clusters presented by Pang-Ning Tan [61].

Well-Separated: A cluster is a set of objects in which each object is closer (or more similar) to every other object in the cluster than to any object not in the cluster. To show all the objects in the cluster, they must be sufficiently similar to one another. Sometimes threshold is used to show this. In well separated clusters the distance between any two points or objects in different groups is larger than the distance between any two points or objects within a group. Well separated clusters can have any shape and need not be globular.

Prototype- Based: A cluster is a set of objects in which each object is more similar to the prototype that defines the clusters than to the prototype of any other cluster. The prototype of cluster is often centroid i.e. the average (mean) of all the points in the cluster or medoid, i.e. the most representative point of a cluster.

Graph-Based: If the data is represented as a graph, then nodes of graph are objects and links between nodes represent connection among objects. In this case a cluster can be defined as a connected component i.e. a group of objects that are connected to one another but have no connections to objects outside the group.

Density-Based: A cluster is dense region of objects that is surrounded by a region of low density. A density based definition of a cluster is often used when the clusters are irregular.

Shared –Property (Conceptual Clusters): A cluster is a set of objects that share some property. A clustering algorithm would need a very specific concept of a cluster to successfully detect these clusters. The process of finding such clusters is called conceptual clustering.

Measures of Similarity and Dissimilarity in Clusters

Pang-Ning Tan [61] described, the most important factor in clustering process is similarity measure and dissimilarity measure. Some transformations can be used to convert a similarity to dissimilarity or vice versa. The similarity between two objects is a numerical measure of the degree to which the two objects are alike.

Cluster algorithms group similar entities together. In order to talk about the similarity of entities and say things like “entity ‘a’ is more similar to entity ‘b’ than it is to entity ‘c’ ” we need some kind of measure of similarity. Similarity measures determine how similar a pair of classes is. Similarity of classes can be calculated by variety of ways and choosing similarity measure is influence the result than the algorithm.

T.A. Wiggerts, [93] described, a similarity measure always yields a value between 0 and 1. Two entities are more similar when their similarity measure comes closer to 1. Often dissimilarity measures are used. From these measures, similarity measures can easily be computed as follows: sim (i, j) =1-dis (i, j).

Clustering is used for grouping the similar things or entities. Work with groupings is strongly connected with the theory of similarity and dissimilarity. One characteristic of a grouping might be that all things within one group are similar and all pairs of elements of different groups are dissimilar. In more detail knowing that two given things are similar is not enough: There are “degrees of similarity”. The same holds for dissimilarity.

Clustering applications typically employ three types of similarity measures, namely, distance measures, correlation coefficients and association coefficients. Distance measures numerically describe how far apart entities are, and these are typically used when features are continuous. Correlation coefficients are usually used for correlating continuous features. Association coefficients are usually applied to binary features.

Onaiza Maqbool and Haroon A. Babri [59] presented some well-known Distance and Similarity measures as given below.

Association Coefficients: Association coefficients are applied to calculate similarity when the features are binary. To illustrate how these coefficients are calculated, assume two entities E1 and E2, represented by feature vectors indicating the presence or absence of a feature. The similarity between E1 and E2 can be compactly represented by a table as shown below:

E2

		1	0
E1	1	A	B
	0	C	D

In the above table ‘a’ represents the count of features present in both E1 and E2, ‘b’ represents the total number of features present in E1 but absent in E2, ‘c’ represents the total number of features present in E2 but absent in E1, and ‘d ’ represents the number of features that are absent in both E1 and E2. It is worth noting that in the software domain, typically d will be much larger than a, b, and c since the feature vector associated with each entity is likely to be sparse.

Let cxy be the resemblance coefficient for components x and y. Some examples are given by Chung-Horng Lung [12].

- Jaccard Coefficient: cxy = a / (a + b + c)
- Russel and Rao Coeffient: cxy = a / (a + b + c + d)
- Simple Matching Coefficient: cxy = (a + d) / (a + b + c + d)
- Sokal and Sneath: cxy = 2a / [2(a + d) + b + c]
- Sorrenson Coefficient: cxy = 2a / (2a + b + c]
- Yule Coefficient: cxy = (ad - bc) / (ad + bc)

The following association coefficients can then be defined.

Similarity measure (S)	Formula
Simple matching coefficient	$\frac{(a+d)}{(a+b+c+d)}$
Jaccard coefficient	$\frac{a}{(a+b+c)}$
Sorenson-Dice	$\frac{2a}{(2a+b+c)}$
Rogers and Tanimoto	$\frac{(a+d)}{(a+2(b+c)+d)}$
Sokal and Sneath	$\frac{a}{(a+2(b+c))}$
Gower and Legendre	$\frac{(a+d)}{(a+1/2(b+c)+d)}$

Table III.a: Well- known Association Coefficients

In table III.a a represents the number of features that are “1” in both entities, d represents the number of features that are “0” in both entities, whereas b and c represent the features that are “1” in one entity and “0” in the other.

Distance Measures: The distance measures calculate the dissimilarity between entities. The larger the distance, the lesser is the similarity between the entities. The measure is zero if and only if the entities have the same score on all features. Some of the most popular distance measures are the (squared) Euclidean Distance, Canberra Distance, Murkowski Distance.

Distance measure (D)	Formula
Minkowski Distance (r = 1, City block distance) r = 2, Euclidean distance)	$\left(\sum_{i=1}^{s}\left\|x_i - y_i\right\|^{r}\right)^{1/r}$
Canberra Distance	$\sum_{i=1}^{s}\left\|x_i - y_i\right\| / \left(\left\|x_i\right\| + \left\|y_i\right\|\right)$
Bray-Curtis Distance	$\frac{\sum_{i=1}^{s}\left\|x_i - y_i\right\|}{\sum_{i=1}^{s}(x_i + y_i)}$
Chord Distance	$\sqrt{2\left(1 - \frac{\sum_{i=1}^{s} x_i y_i}{\sqrt{\sum_{i=1}^{s} x_i^2 \sum_{i=1}^{s} y_i^2}}\right)}$
Hellinger Distance	$\sqrt{\sum_{i=1}^{s}\left[\sqrt{\frac{x_i}{x_+}} - \sqrt{\frac{y_i}{y_+}}\right]^2}$ where $x_+ = \sum_{i=1}^{s} x_i$

Table III b: Well-known Distance measures

In table III b, x and y represents points in the Euclidian space Rs

Correlation Coefficients: Correlation coefficients are used to correlate features. They are applied to the correlation of entities as well although it makes no statistical sense to obtain mean value across different feature types rather than across entities. The well - known Pearson product moment correlation coefficient for binary features reduces to:

P = (ad-bc) / √ (a+b) (c+d) (a+c) (b+d)

According to O. Maqbool [58], in the case of software, since normally d is much larger than a, b and c, the above formula can be written as:

P = a / √ (a+b) (a+c)

The value of a correlation coefficient lies in the range from -1 to 1. A value of 0 means that the two entities are not related at all.

Probabilistic Measures: According to T.A. Wiggerts [93] probabilistic measures are based on the idea that agreement on rare features contributes more to the similarity between two entities than agreement on features which are frequently present. So, probabilistic coefficients take into account the distribution of the frequencies of the features present over the set of entities. When this distribution is known, for each feature a measure of information or entropy can be computed. The entropy quantifies the disorder, variance, confusion or surprisal. The two (sets of) entities which provide the least information gain (change of entropy) when combined have the highest similarity.

Linkage Methods: During clustering the similarity between the newly formed and existing components should be iteratively recalculated. For this recalculation various linkage methods are available. Some of the well-known linkage methods are presented below in table III c.

Algorithm	**Cluster Similarity**
Single Linkage (SLA)	$sim(E_i,E_{mo})=Max(sim(E_i,E_m), sim(E_i,E_o))$
Complete Linkage (CLA)	$sim(E_i,E_{mo})=Min(sim(E_i,E_m),sim(E_i,E_o))$
Weighted Average Linkage (WLA)	$sim(E_i,E_{mo})=1/2(sim(E_i,E_m))+1/2(sim(E_i,E_o))$
Unweighted Average Linkage (ULA)	$sim(E_i,E_{mo})=(sim(E_i,E_m)*size(E_m)+sim(E_i,E_o)*size(E_o))/(size(E_m)+size(E_o))$

Table III c: Well known Hierarchical Linkage Methods

These four linkage methods presented in Table III c determine similarity between a newly formed cluster and existing entities by using given cluster similarity formulas. In table III c Ei, Em, and Eo represent entities and Emo represents the cluster formed by merging entities Em and Eo.

Dissimilarity Measure: According to Pang- Ning Tan [61], the dissimilarity between two objects is a numerical measure of the degree to which the two objects are different. Dissimilarities are lower for more similar pair of objects. Often the term distance is used as synonym for dissimilarity. The common interval for dissimilarity is [0, 1] but can range from 0 to ∞.

Dissimilarity Measures are used to find dissimilar pairs of objects in X. The dissimilarity coefficient, dij, is small when objects i and j are alike, otherwise, dij becomes larger. A dissimilarity measure must satisfy the following conditions:

• $0 \leq dij \leq 1$

• $dii = 0$

• $dij = dji$

Typically, distance functions are used to measure continuous features, while similarity measures are more important for qualitative features. Selection of different measures is problem dependent. For binary features, the similarity measure is commonly used. Let us assume that a number of parameters with two binary indexes are used for counting features in two objects. For example, n00 andn11 denote the number of simultaneous absence and presence of features in two objects respectively, and n01 and n10 count the features presented only in one object.

Appendix – II (c)

Research Paper Repository

a. Component evaluation and component interface identification from object oriented System by Shivani Budhkar, Dr. Arpita Gopal, in International Journal of Advanced Research in Computer Science, ISSN No. 0976-5697, Volume 3, No. 4, July- August 2012

b. Component identification from existing object oriented system using Hierarchical clustering by Shivani Budhkar, Dr. Arpita Gopal, in IOSR Journal of Engineering, ISSN: 2250-3021, May. 2012, Vol. 2(5) pp: 1064-1068

c. Component based software architecture recovery from object oriented system using existing dependencies among classes by Shivani Budhkar, Dr. Arpita Gopal,in International Journal of Computational Intelligence Techniques ISSN: 0976-0466 & E-ISSN: 0976-0474, Volume 3, Issue 1, 2012, pp.-56-59.

d. Reverse Engineering Java Code to Class Diagram: An Experience Report, by Shivani Budhkar, Dr. Arpita Gopal, in International Journal of Computer Applications (0975 – 8887) Volume 29– No.6, September 2011, pp. 36-43

e. Component interactions from software architecture recovery by Shivani Budhkar, Dr. Arpita Gopal, in International Journal of Computer Science and Communication Vol. 2, No. 1, January-June 2011, pp. 149-15

f. Extraction of Connector Classes from Object –Oriented System while recovering Software Architecture by Shivani Budhkar, Dr. Arpita Gopal, in IEEE International Advance Computing Conference (IACC 2009) Patiala, India, 6–7March 2009, pp.1826-1828