International Journal of Software Engineering and Its Applications
Vol. 9, No. 1 (2015), pp. 35-52
http://dx.doi.org/10.14257/ijseia.2015.9.1.04

Reengineering Techniques for Object Oriented Legacy Systems

Aman Jatain and Deepti Gaur

[1]*Department of Computer Science and Information Technology, ITM University, Gurgaon*
[2]*Department of Computer Science and Information Technology, ITM University, Gurgaon*
[1]*amanjatainsingh@gmail.com*, [2]*deeptigaur@itmindia.edu*

Abstract

Today's software development is defined by continuous evolution of software products. These products are regularly updated during their usage. In most of the cases systems grow inevitably by adding new features or by changing the system architecture due to new technologies or business plans. It is more than a decade; objects oriented paradigm is adopted as the most efficient passage to build flexible software, and promptly supported by industry. Though, the benefits of object oriented paradigm are supported by many, but its usage does not necessarily result in general, adaptable systems. These huge systems are often suffering from improper use of object oriented techniques, like inheritance and the lack of object oriented methods being regulated towards the building of families of systems instead of developing single applications. These growing technologies make the systems more difficult to maintain and improve. So, there is growing demand for reengineering of object-based systems. The main intention of this paper is to discover important research directions in the area of reengineering of object oriented legacy systems, which necessitate further attention in order to build more effective and efficient reengineering technique for these systems. The paper first discusses the state of art in reengineering of legacy system and its need. Paper also discuss the benefits of component based system over object oriented system and later outlines the techniques for reengineering of object oriented legacy system. In this paper we presented statistical analysis based on more than a decade data.

Keywords: *Forward, Legacy, Object Oriented Reengineering, Reverse, Transformation*

1. Introduction

In today's software industries reengineering of legacy systems has become an integral activity. In the last few years, most of the reengineering efforts were concentrated on systems written in conventional programming languages such as FORTRAN, C and COBOL [29]. Many software systems have been maintained for many years, these are "legacy systems" that are vital for the organization, but often difficult to understand and maintain [45]. Ian Somerville defines the legacy systems as "older legacy system" that remain vital to an organization. The business values of legacy systems have been decreased due to the lack of standardization, inflexibility, rapidly changing technology and lack of distributed architecture. Despite of these limitations, the importance of legacy systems cannot be ignored because some of their functions are too valuable to be removed and expensive to reproduce [28]. Earlier objects oriented paradigm is adopted as the most efficient passage to build flexible software, and promptly supported by industry. Though, the benefits of object oriented paradigm are supported by many, but its usage does not necessarily result in general, adaptable systems. These huge systems are often suffering from improper use of object oriented techniques, like inheritance and the lack of object oriented methods being regulated towards the building of families of systems

ISSN: 1738-9984 IJSEIA

instead of developing single applications. So, these systems need to be reengineered in order to fulfill the rapidly changing requirements. This recent evolution is caused by the fact that the same application of object-oriented techniques is not appreciated to deliver flexible and adaptable systems. The consequence is a new generation of inadaptable legacy systems. Early adopter of the object oriented paradigm now face the issues related to transformation of these object oriented legacy systems into the more reusable framework and components.

Component based software development (CBSD) promotes the acquisition, adaptation, and integration of reusable software components. These techniques allow the developer to rapidly develop and deploy complex software systems with minimum effort and resource cost [40]. A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third part [27l. The software communities have recently paid more attention towards the component based software development to upgrade the quality and to increase the productivity. Instead of re-developing the business critical systems from scratch with in time and resource constraints, the best option is to make the legacy system adaptable to new technologies. Reengineering is defined as; [31]

- Re-developing legacy systems to improve their maintainability.
- Re–documenting, organising and restructuring the system.
- Translating the system to a more modern programming language and modifying and updating the structure and values of the system's data without changing the functionality of the software and system architecture.

Software reengineering has become an important sub discipline within computer science from past decade. Bigger organizations increasingly automate important and critical tasks, which make their business highly dependent on information systems. But in many cases, these systems have been sustained for many years, these are legacy systems that are vital for the organization, but they are often hard to understand and maintain. In terms of time and budget the redevelopment of these systems from scratch is not a good idea. So, other option is reengineering.

For reengineering of a system, one should study about history of reengineering, phases of reengineering, tools and techniques of reengineering. Cost of reengineering the system and reengineering experiences has to be examined to design a complete reengineering model. Different researchers have different meaning for reengineering. "Re-engineering is the examination, analysis and alteration of an existing software system to reconstitute it in a new form, and the subsequent implementation of the new form" [14], Dr. Linda (1996). Chikofsky and cross (1993) in [17], gave the concept of reengineering as the examination and modification of a subject system to reconstruct it in a new form and the successive implementation of the new form.

Daniel *et al.* (2002) in [12] defined reengineering as "Reengineering concerns the examination of the design and implementation of an existing legacy system and applying different techniques and methods to redesign and reshape that system into hopefully better and more suitable software". Software Reengineering has primarily three phases: (1) Reverse Engineering (2) transformation of architecture and (3) Forward Engineering. Reverse engineering is a part of the reengineering life cycle. It is the process of analyzing a subject system to identify the system's components and their interrelationships and create representation of the system in another form or at a higher level of abstraction [14]. Reverse engineering is often a prior step of reengineering. Forward engineering on the other hand is the traditional process of moving to high – level of abstractions and logical implementation [17]. Figure 3.1 illustrates this idea:

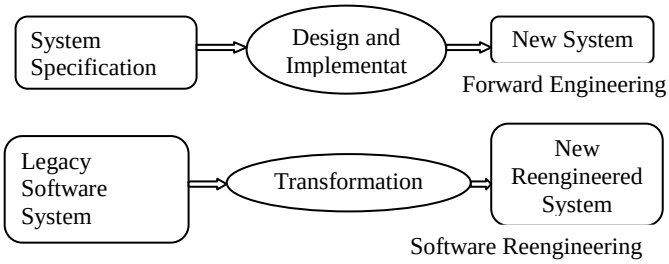

Figure 3.1. Forward Engineering and Reengineering

These days reengineering of object oriented system is upcoming field of research because software professional are now dealing with large amount of industrial source code, developed using object oriented methodologies of the late 80s and early 90s. While dealing with these systems, a lot of issues occur, effectively preventing software professional from satisfying the changing customers requirements. Though the object oriented concept is comparatively new, there are many object oriented legacy systems. The object oriented paradigm concept has been accepted by many software enterprises but now with the existence of millions line of code in object oriented legacy systems, reengineering of these systems is a speeding area of research. Early adopter of the object oriented paradigm now facing the issues related to transformation of these object oriented legacy systems into the more reusable framework and components. So, tools or methodologies support is required to deal with the large and poorly documented object oriented programs. The reengineering of object oriented system has been gaining attentions to reduce development costs and to facilitate the software maintenance.

Reengineering of software system has become popular in the early 1990s and at that time it was not fully understood. In the twenty first century reengineering of software system is an effective instrument for organization to meet changing requirements of their clients and to increase productivity and saving cost by developing reusable and flexible system components. The need for reengineering of object-oriented system has been realized by two of the leading European companies, namely Daimler-Benz and Nokia. These two companies started a research project named FAMOOS together with the University of Berne, Informatik, SEMA Spain and Take5es to investigate tools and technique for dealing with object oriented legacy systems. All the techniques in FAMOOS project are verified by case studies collected from six industries [29].

In order to satisfy the changing customer requirements, the industries need to reengineer these bulky and inflexible object oriented legacy systems to adaptable frameworks and libraries of small, understandable software components. Such frameworks will allow a greater flexibility and deal efficiently with changing customers requirements of different perception, as well as an easier integration of new changing requirements [4].

This paper is organized into four remaining sections. Section 2 provides need of reengineering and discusses some benefits of component based system over object oriented system In Section 3 there are three phase: in the first phase research methodology is discussed and three main questions are posed. In second phase the search is performed by using different queries related to reengineering techniques and in the third phase report is validated by researchers. The paper conclusions and future work are presented in Section 4.

2. Reengineering of Object Oriented Legacy Systems as Compare to Component Based System

Mainly the need of reengineering legacy system is driven by three factors [46]:

- Expansion of the system's functionality;
- Improved maintainability of the system using latest tools and techniques; and
- Reduction of operational costs by increasing reuse potential of the system.

The need of reengineering has largely increased as legacy system become outdated in terms of their architecture, the platform and the support. For recovering and reusing of the useful legacy system assists, software reengineering is an important activity. Legacy system reengineering is a vital activity in software reuse because it helps in extract important operations and reusable components from the legacy system [30]. Software reengineering reduces high software maintenance costs and establishes a maintainable base for future software evolution. Reengineering of object oriented legacy system aims to enhance the overall quality of software systems by making the functionality of these systems easier to use and reusable, to prepare a legacy system for future change and adaptable to new application areas.

To achieve the required level of adaptability is to decompose the system into a set of components. "Components are binary units of independent production, acquisition, and deployment that interact to form a functioning system", [10]. Component based software engineering (CBSE) is a paradigm that aims at constructing and designing systems using a predefined set of software components explicitly created for reuse [7]. In object oriented paradigm source code is reused in the form of objects, and the developers reuse these objects by using various mechanisms such as inheritance and polymorphism. The principle is same with CBSE but whole software component is reused instead of objects. There are various reasons given in literature, why software systems made up of components are better than object oriented system while dealing with changing business needs.

- Component based programming basically emphases on constructing and packaging robust, scalable and adaptable components to permit developers or reusers to build their system quickly and efficiently, which promotes software reuse and in turn increase software productivity [20].
- In component framework components are developed independently and loaded dynamically as compare to classes that are linked together.
- Components framework are black box framework *i.e.*, framework that can be reused without modification in their source code and can be extended through composition as compare to object oriented framework.
- In component based development, reuse of component can reduce the development and maintenance cost of new systems, while in object oriented programs cannot be reused effectively in the components based development even if they contain reusable functions [26].
- OO classes usually are mutually dependent making difficult to reuse parts of existing OO programs composed of classes, so it is necessary to transform a part of an existing OO program into component that has no dependence on element outside itself [26].
- Component framework increases the flexibility and prepares the legacy system for future change [5].
- Distributed systems technology is more and more based on the use of component technology [15].
- Components are observed as being more coarse-grained as compared to objects and provide high level of abstraction [22].
- Large size and complexity of object oriented legacy system leads to huge and complicated collection of classes and objects that are difficult to handle and understand. These systems can be improved by advanced means of structuring,

describing and developing. Component framework is a possible approach to solve these problems [15].

Holger, Markus and Oliver *et al.* [29] in their Object-Oriented Reengineering Handbook which is based on the famous 'FAMOOS' project mentioned that goal and motivation of reengineering of object oriented legacy system into CBD is to unwrap the software system into subsystems i.e components that can be tested , delivered and marketed separately. Pressman in 1997 and O'Donnell in 2001 discussed that maintenance of legacy systems is a very difficult task for every organization Almost 80% of software development costs are used to maintain the systems after they have been implemented. Insigma Technology Co. 2001, used to provide IT services in legacy reengineering. It reengineered several financial industry legacy systems and one of its client 'State Street Corporation' said that "reengineering their project compared to the cost of the system replacement saved millions of dollars".

O'Donnell [42] in 2001 briefed that, one major advantage of a component is its plug and play feature which allows easy composition, selection and adaptation of components rather than implementing the application from scratch. Dr. Pami Bahsoon, School of computer science, the University of Bermingham supported the concept of CBSE by presenting the fact that it emerged from the failure of object oriented development to provide effective reuse because single object classes are too detailed and specific. Components are more abstract that object classes and can be considered as standalone service providers.

Simon and Salah et al. [50] in 2011 described that object oriented systems are complex because they contain a large collection of classes with varied dependencies while component is assumed to provide modeling elements well suited to high level abstraction and organized structure of complex software. **In 2012, Shivani and Arpita [48]** discussed than object oriented development had not provided pervasive reuse and object technology has not shown substantial development of distributed system while component based technology is considered to be more suited for distributed system development due to its granularity and reusability. **Atish P. Sinha et al. [8] in 2013** provided a empirical study that compares IT professional's perception about the ease of reuse in modeling business system from professional perceive components to be much easier to reuse than objects. **In [1] Ai- Ping and Zheng et al. in 2013** discussed that component has higher independence and integrity and self description as compare to traditional software entity such as object.

3. Research Methodology

Brereton *et al.*, in 2007 presented a systematic review where they have presented the reviews in software engineering. In this paper there is an adopted form of this systematic review. The review process has three phases. In the first phase of review, the following questions are posed:

- **Question1:** What methodologies have been developed to reengineer a legacy system?
- **Question2:** What is the applicability of these techniques?
- **Question3:** What are the main drivers to reengineer the legacy system and challenges faced during the process?
- **Question4:** What are the benefits of reengineering legacy system?

After defining the above questions, a review protocol is introduced. The review protocol includes the sources used in the review process, the time duration and the research criteria used. The review protocol is shown in Table 3.1. The sources used for the review are: IEEE Explore, Google Scholar, Science Direct, Springer Library and ACM digital library. The search focuses on the years 1999 to 2013.

Table 3.1. Review Protocol

Year	Sources	Keywords
1999-2013	IEEE Explore, Google Scholar, Science Direct, Springer Library and ACM digital library	Software Reengineering, Legacy Systems, Object Oriented, Architecture, Component,

In the second phase of the review, the search is performed using different queries related to the software reengineering techniques. A paper is included if it contains either an architecture, model framework or source code. To answer the posed questions above, data is extracted from the papers. A table contains the following column: Author, Year, Methodologies, Tool and the Target System. This table helped to organize the extracted information with the aim of research findings in this area. Finally in the third phase of the review, the report is validated by researchers.

3.1. Results

The review results are presented in this section. A year-wise picture of the results is presented in Table 3.2

Table 3.2. Year Wise Search Result

Year	No. of Publication
1999	1
2000	2
2001	0
2002	0
2003	2
2004	2
2005	2
2006	0
2007	0
2008	2
2009	1
2010	1
2011	1
2012	2
2013	2
Total	20

Question 1: What methodologies have been developed to reengineer a legacy system? The results of the review shown in Table 3.3 summarized the proposed approaches to reengineer the legacy system. The results are categorized into: (1) Framework, where a

framework is presented for reengineering legacy system. (2) Models (3) Refactoring (4) Process (5) Clustering Based Approach (6) Metrics. The results are presented in Table 3.4.

Table 3.3: Proposed Approaches for Reengineering

Year	Author	Methodologies
1999	Holger, Markus and Oliver	Framework
2000	Serge, Stephan	Framework
2000	Arie van, Ben	Process
2002	Miguel, Doris	Clustering Algorithm
2003	Eunjoo Lee, Byungjeong	Process
2003	Michele Lanza	Metrics
2004	Anders Hessellund	Refactoring
2004	Eda, Francesca	Refactoring
2005	Hironori and Yoshiaki	Refactoring
2005	Dharmanlingam	Process
2008	Sylvain Chardigny and Abdelhak	Process
2008	Hassan, Ameur	Framework
2009	S.K and K.S Mishra	Model
2010	Alae-Eddine, A.Djamel	Process
2011	Jian Feng and Heung Seok	Clustering Algorithm
2012	Simon, Salah	Process
2012	Shivani and Arpita	Clustering Algorithm
2012	Sudhakar, P. and P. Sakthivel	Model
2013	Han Li, He Guo	Rule Based Model
2013	Ai- Ping and Zheng	Reuse Model

Table 3.4: Categories wise results of Question1

Question	Category	No. of papers
What methodologies have been developed to reengineer a legacy system?	Framework	3
	Models	4
	Refactoring	3
	Process	6
	Clustering Based Approach	3
	Metrics	1
	Total	20

The results (Figure 3.2) show that most common proposed methodology was to develop a process for reengineering (30%). **Arie van, Ben (2000) *et al.*** in [5] presented a three step approach: find components, global restructuring, and renovate per component for software renovation of legacy system. In this paper author provided a comprehensive view of software renovation and emphasized that aim of software renovation is to prepare the legacy system for future change. Author also discussed that main technique to arrive at the required level of flexibility is to decompose the system into a set of components *i.e.*, component based software development. The aim of software modernization was to understand, transform and regenerate a legacy system so that system is compatible with new business objectives and new technological developments. The three technical approaches presented in [5] for software renovation were:

- **Analysis of Legacy Sources**: During analysis the sources of the legacy system were inspected and extracted information reveals legacy systems structure, purpose and architecture.
- **Transformation of Legacy Sources**: In transformation the sources of legacy system were systematically restructured and enhanced.

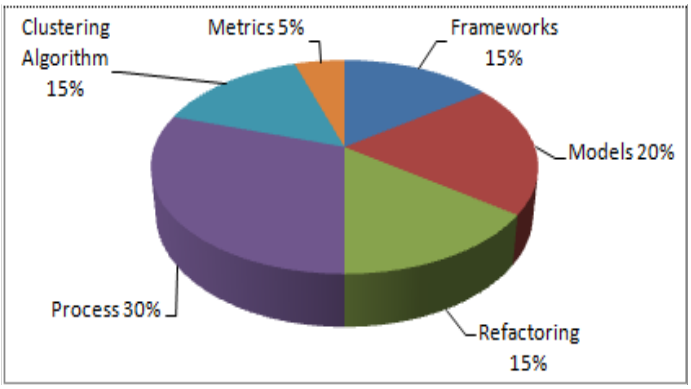

Figure 3.2. Proposed Methodologies for Reengineering

Eunjoo Lee, Byungjeong *et al.* [18] (2003) worked on a process to reengineer an object – oriented legacy system into a component based system. The reengineering process is consisting of two steps: First basic components are created using composition and inheritance relationship between constituent classes that are determined my examining the program source code. This resulted into intermediate component based system. In second step intermediate component are refined using three metric proposed by the authors to assess the software quality. The component metrics proposed are: component complexity metric, connectivity strength and cohesion metric and then the proposed model is applied to an ATM system written in C++.

Dharmanlingam *et al.* [13] (2005) emphasized on the use of reverse engineering to identify or extract components from object oriented systems. The assumption made by author in the proposed method is that reusable classes have some quality attributes e.g. functional usefulness, readability, testability etc. and these quality attributes are mapped to metrics. In the paper the domain specific classes are classified based on the metrics derived. The main behind the proposed approach is reduce the amount of data the human expert has to review in order to identify domain specific classes.

Sylvain Chardigny and Abdelhak *et al.* [53] (2008) proposed a technique of component based architecture recovery from object oriented system. The proposed approach is quasi-automatic process based on semantic and structural characteristics of software architecture concept to recover architecture. Authors named the proposed approach as ROMANTIC. Architecture extraction is the reverse process of the design. The extraction process consisted of two steps. The first step is the defining of a correspondence model between object concepts and architectural ones. The correspondence model is the first principle which is derived by the second principle which defines a set of rules for the extraction process. After preparing the correspondence model, architecture is considered as a partition of the system classes. In the partition each element represents a component. This second principle defined the relevance of architecture and method to instantiate the previous model

Alae-Eddine, A.Djamel *et al.* (2010), in [2] proposed another technique of component based architecture recovery from object oriented systems using a semi automatic exploration process and for identification of architectural component 'Relational Concept Analysis' (RCA) approach is proposed. RCA approach is extension of the ROMANTIC approach discussed in [53]. RCA is based on the identified source code entities and relationship between them. In this approach architectural components are extracted from concepts and these concepts are derived by exploiting all existing dependency relations between classes of the OO system. The feasibility of the RCA is evaluated on Java

Software. In the paper to identify the relationship between classes various methods are analyzed to calculate the metrics and these relations are used in RCA model to generate the lattice of concepts. The obtained lattices are used to extract architecture and select components according to some grouping criteria and navigating in the lattice.

Simon, Salah *et al.* [50] in 2012 tried to automate the process of transformation of object oriented application to an component oriented application. Two steps are discussed by authors that are necessary to produce component oriented architecture from an object oriented system *i.e.*, 1) identification of components, 2) extraction of components interfaces (architectural view) and in last step transformation of the application using a concrete component model (OSGi). The proposed method is also implemented on a case study to assess the feasibility.

15% of the approaches proposed for reengineering of object oriented legacy system are framework based. Further frameworks are used by researcher to develop a tool for reengineering. **Oscar [44]** discussed the FAMOOS (ESPRIT program Project no. 21975: Framework based approach for understanding object oriented software evolution) project initiated in November 1996 and launched in December 1999. The objective of the work was to develop toll and techniques to migrate object oriented legacy system

Holger, Markus and Oliver et al. [29] in their Object-Oriented Reengineering Handbook which is based on the famous 'FAMOOS' project mentioned that goal and motivation of reengineering of object oriented legacy system into CBD is to separate out the software systems into modules or subsystems that can be tested, delivered and marketed separately. Pressman in 1997 and O'Donnell in 2001 discussed that maintenance of legacy systems is a very difficult task for every organization Almost 80% of software development costs are used to maintain the systems after they have been implemented. Insigma Technology Co. 2001, used to provide IT services in legacy reengineering. It reengineered several financial industry legacy systems and one of its client 'State Street Corporation' said that "reengineering their project compared to the cost of the system replacement saved millions of dollars".

Serge, Stephan *et al.* [47] in 2000 proposed a framework based approach for reengineering object oriented systems using restructuring transformation. This paper was position paper on FAMOOS project. They examined five case studies using the FAMOOS project strategies.

After analyzing these five case studies they felt that a reengineering of an object oriented system must support two levels of restructuring *i.e.*, 1) low level restructuring clean up the source code, repairing and refining the structures and dependencies; and 2) high level restructuring resolve architectural problem. A tool named MOOSE is also developed in this paper and MOOSE helps in recovering multiple views and combining the information gathered from different views.

Hassan, Ameur *et al.* (2008) in [25] presented a framework that help in generation of component based software from object oriented systems. The components are generated by using various steps: 1) extraction of class diagram; 2) the analysis phase; 3) graph generation and processing phase and 4) the component generation phase. In the first phase extract class diagram are extracted from java code using UML models and then the exported into XMI file using 'ArgoUML' UML tool. Then in next step class diagrams are analyzed to extract elements and relationship between them to calculate the dependency weights. Elements acts as nodes and arcs represent the relationship between classes. In third step directed weight graph is constructed by supplying information obtained from step 2. In final phase the clustered graph is processed to identify the components. Each

component is composed of set of elements that forms a cluster. After implemented the proposed approach author felt that migration of object oriented system into components decreases the complexity of the system.

Some of the results of the review (15%) consisted of reengineering models and rule based models. These include **S K Mishra (2009)** in which [51] a model named as CORE (Component Oriented Reverse Engineering) is proposed for identification of reusable components. The model uses reverse engineering methods to generate components from object oriented systems. CORE model makes use of repository to manage and store the tested components and reconstituting the new system using reusable components. OOAD models are used to generate use case diagram, interaction diagram and class diagram. For notification of the reusable components in a more effective way concepts of CRUD matrix and class clustering is also implemented in the proposed work.

Sudhakar, P. and P. Sakthivel (2012), in [52] proposed 'Reengineering Legacy to Modern' (RL2m) model with One Time Checker (OTC). The model is used to extract source code form legacy systems. The steps involved in the model are as follows: 1) Program Analysis 2) Identification of the constructs from the legacy system 3)slicing of the code 4) Mapping source and destination program to create template 5) Creating wrapper 6)execution 7)Integrated to VLSI application. In the model the dynamic slicing is used as debugging technique for system extraction process. Dynamic slicing is a program slicing technique where the source code is decomposed to produce the slices. An algorithmic approach is followed by author for dynamic slicing. Various challenged faced (like how to deliver a good translation methods and attributes, and making sure that the new system passes the regression test) during reengineering process is also highlighted in the paper and as a solution of the challenges faced, a new method OTC (One Time checker) is also discussed. OTC is used to analyze the target before the compilation and execution of the system after reengineering process. RL2M model is also evaluated for program slicing by conducting an experiment.

Ai- Ping and Zheng *et al.* (2013) in [1] provided a experience on legacy system reengineering based on component reuse in a insurance company in social security system located in China. They proposed component based hierarchical architecture reuse model and the model is then applied on old age insurance MIS under social security information service platform. The new MIS system was more effective, convenient and component performance was also improved.

Some of the researchers have also proposed refactoring technique (15%) to renovate the legacy system. According to **Anders Hessellund (2004)** in [3] object oriented systems are slowly beginning to surface in domain reengineering. Author explained that a proper reengineering requires an intimate knowledge of the existing system. So in this paper refactoring technique is discussed to reengineer the legacy object oriented legacy systems. Refactoring is defined as "the change made to internal structure of software to make it easier to understand and cheaper to modify without changing its observable behavior". Author explained that refactoring is implemented in small, incremental steps. First one must build a set of tests that covers the entire path through the code and check if the code is working and then low level refactoring such as Extract Method and Move Method till the well maintained hierarchy is established. **Eda, Francesca *et al.*, [16] (2004)** discusses their experience of reorganizing or refactoring a monolithic piece of software in component based manner in [16]. To obtain useful component architecture, author followed the UML Components methodology described by Cheesman and Daniels in 2000 in their book. Eda applied the UML Component methodology on JAIN MAP API case study and drives suitable component architecture. Author also concluded that by

using the proposed refactoring technique, the software obtained is more flexible, testable and manageable.

Hironori and Yoshiaki (2005) in [26] proposed a refactoring technique to identify and retrieve reusable components from OO systems. The technique helps to perform refactoring automatically and extract JavaBeans components from java program. The proposed method targeted java language as the object oriented language and the JavaBeans as the fundamental component architecture. Extraction of components is done in two steps: in first step to represent the relations between classes CRG (Class Relation Graph) is constructed and in next step clustering extraction algorithm is implemented to detect clusters that are candidate of components.

Review results show that 15% of the studies used clustering algorithm to modernize the legacy system. **In [39] Miguel, Doris *et al.*, (2002)** discussed that object oriented systems have many problems similar to legacy systems. These problems include extensive maintenance activities, lack of tool support and improper training. In this paper to reengineer the object oriented system first basic object oriented constructs, their physical dependences and their logical dependences are identified because object oriented systems have complex relationship and interactions among activities. After this step the system is decomposed into high cohesive and low coupled subsystems *i.e.*, components that are suitable for distribution. Decomposition into subsystem or components consisted of five steps:

1. A set of related object based constructs such as objects, classes attribute, and methods are generated.
2. By using the object oriented metric techniques a metrics set is generated. These metric sets contain groups of related classes or classes that have some relationship or dependence.
3. From the generated metric sets in step 2, interaction matrices are defined.
4. Data mining algorithm or association rules are applied to obtain the support and confidence of the association rules.
5. Then a hierarchical clustering algorithm is used over the association coefficients to produce a hierarchical decomposition of the system.

Jian Feng and Heung Seok (2011) in [32] examined object oriented legacy system with a purpose of migrating it to component based system. For the migration process various techniques of agglomerative clustering algorithms are employed. Clustering is done using similarity measure, linkage methods and weighting scheme. They named the whole method as CIETool. It make use of clustering algorithms and various evaluation criteria to automatically extract class dependency information and which helps in identification components. Authors have conducted 18 clustering algorithms, considering similarity measures, weighting schemes and linkage methods to check the feasibility of the tool

Shivani and Arpita (2012) in [49] discussed that most of the object oriented system's architecture is not reliable while dealing with changing technology. To overcome this problem an approach was proposed in this paper to extract component architecture from object oriented system. Architecture is recovered using class dependencies and by implementing clustering algorithm. It is a three steps approach and steps included are: 1) Identification of dependencies among classes; 2) Identification of components using hierarchical clustering: 3) Identification of interfaces for binding components together. The developed tool accepts object oriented java code and transforms it into component based system. Classes are clustered by identifying inheritance coupling, composition coupling and method coupling.

Metrics presents 5% of the results. In [38] **Michele Lanza (2003)** in his thesis performed reverse engineering of an object oriented system by combining two techniques: 1) Software Visualization and 2) Software metrics. By using combination of these two techniques a new approach is developed named as Polymetric view. The polymetric views were used in three different reverse engineering contexts *i.e.*, 1) Coarse- grained software visualization; 2) fine-grained software visualization and 3) evolutionary software visualization. Proposed methodology is applied on several case studies in the paper and author concluded that it helps in reducing the complexity involved in reverse engineering process and also supports opportunistic code reading.

Question2: What is the applicability of various proposed methodologies in literature? The applicability of the proposed methodologies is categorized in two ways. First, the techniques are distinguished according to whether the legacy systems are developed in COBOL, C, C++ or Java. Second the methodologies are categorized according whether the reengineered legacy systems are object oriented system, cloud based system, and aspect oriented system or component system.

Table 3.5. Categories Wise Results of Question 2

Question	Category	No. of papers
What is the applicability of these methodologies?	C++	12
	Java	3
	Cobol	2
	C	1
	Total	18
	Component based system	14
	Object Oriented system	3
	Cloud based system	2
	Aspect oriented system	1
	Total	20

Figure 3.3 shows that 60% of the legacy systems which are reengineered were originally developed in C++ (object oriented language), 15% were developed in JAVA, 10% in COBOL and 5% of the legacy system were developed in C *i.e.*, procedural language.

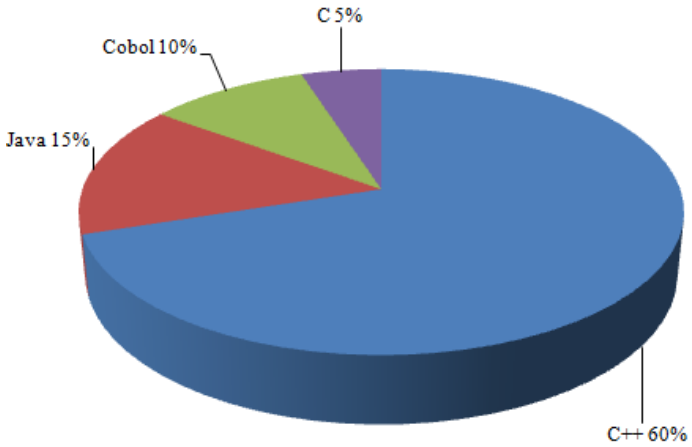

Figure 3.3. Language Based Classification of Legacy System

Figure 3.4 further categorizes the results according to whether the reengineered legacy systems are object oriented system, cloud based system, aspect oriented system or component based system. 70% of the legacy systems are reengineered in component based system, 15% in object oriented system, 10% in cloud based system and 5% in aspect oriented system.

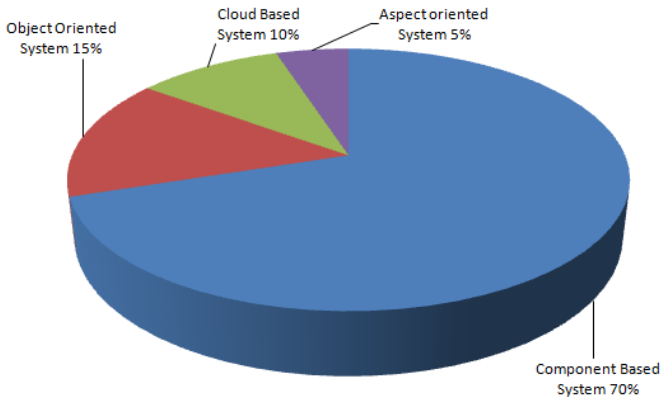

Figure 3.4. Orientation of Reengineered Legacy System

Question3: What are the main drivers to reengineer the legacy system and challenges faced during the process? The identified drivers for reengineering the legacy system include (1) High Maintenance Cost, (2) Hard to adapt new technologies, (3) Lack of Expertise and (4) Increased reuse. Some of the challenges identified during the modernization are (1) Lack of documentation (2) Complex architecture (3) Lack of knowledge (4) Support from organization (6) hard to adapt new technology and (5) Difficult to extract business rules. Table 3.5 provides numbers of papers for each category.

The results of the question regarding drivers for modernization of legacy system shows that in 35% of selected papers researchers discussed that changing technologies is the main driver. High maintenance cost as modernization drivers is discussed in 30% of the papers. In 20% of the papers explain that increased reuse is the main driver and 15% of the selected papers provide fact that lack of expertise is the main driver for modernization as shown in Figure 3.6.

Table 3.6. Categories Wise Result of Question 3

Question	Category	No. of papers
What are the main drivers to reengineer the legacy system	High Maintenance Cost	6
	Changing technology	7
	Lack of expertise	3
	Increased Reuse	4
	Total	20
What are challenges faced during the process?	Lack of documentation	3
	Complex architecture	4
	Lack of knowledge	5
	Support from organization	2
	Hard to adapt new technology	4
	Difficult to extract business rules	2
	Total	20

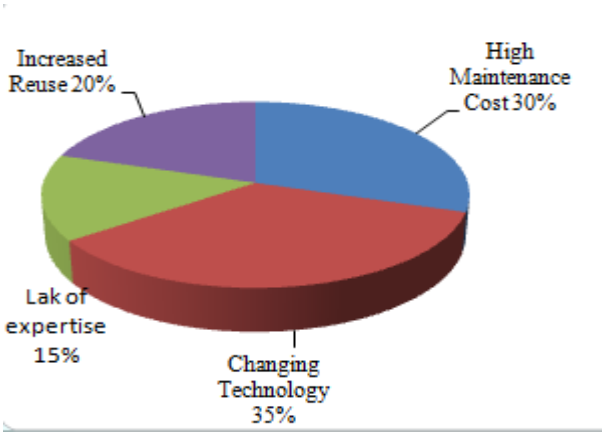

Figure 3.5. Drivers for Modernization

The results associated with the question, what are the challenges faced during modernization? shows that 20% of the papers discusses the complex architecture as a challenge, lack of knowledge is discussed in 20% of the papers, 20% provided fact that adaptability of legacy system is a big challenge, lack of documentation as a challenge is provided by 15% of papers, 10% of the papers discusses that support form organization is one of the challenge and 10% focuses that extraction of business rule is another challenge faced during modernization of legacy system. Statistic of the question is provided in Figure 3.6.

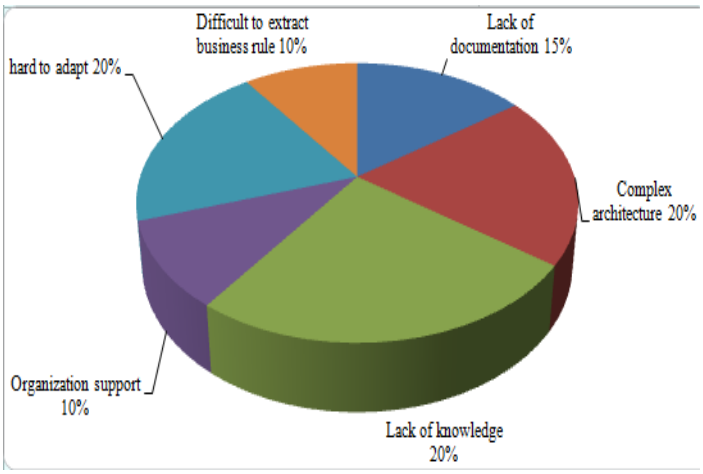

Figure 3.6. Challenges Faced During Modernization

Question4: What are the benefits of reengineering legacy system? Observed benefits of the reengineered legacy system are (1) Reliability (2) Reduced complexity (3) Proven Technology (4) Performance of the system. As shown in Figure 3.7, 40% of the selected paper provided facts that reengineered legacy system are better system technology wise.

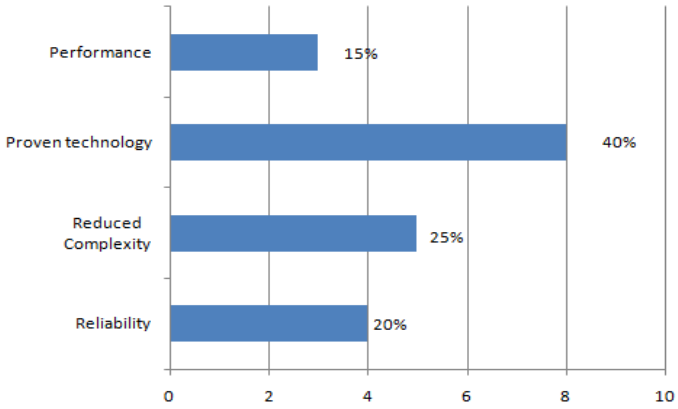

Figure 3.7. Observed Benefits of Reengineered Legacy Systems

The perceived benefits of the reengineering are discussed in some of the selected papers which include: Serge, Stephane, Robb *et al.*, in [47] explained that after reengineering of the legacy systems, flexibility, reliability and adaptability of the system is improved. In [5], Arie, Ben *et al.*, have provided some facts which promises the increased reusability of the system after reengineering. Miguel, Doris *et al.*, [39] have given some evidence (where they validated the result using case studies) that reengineered system has better performance. In [13], Dhrmalingam Ganesan discussed that reuse capability of the legacy system is increased after reengineering using the proposed approach. Philippe Dugerdil in [45] explained that restructuring (which is sub phase of reengineering) improved some of the quality attributes of the legacy system which include performance of the system also. Hassan, Ameur *et al.*, [25] discussed that renovation of legacy system decreases the complexity of the system which in turn reduces the maintenance cost. Simon Allier, Salah *et al.*, [50] validated through a case study that reengineered object oriented system provides better understanding and reusability. In [48], Shivani and Arpita proposed a reengineering technique for object oriented system and discussed that the after applying the proposed reengineering approach, reliability and performance of the system improves.

4. Conclusions and Future Directions

In this paper a systematic review of the work in the area of software reengineering is conducted and review results are presented. The review results shows that majority of the methodologies are process based, where a process is discussed to reengineer a legacy system, no validation or automatic tool is provided and the target legacy system are object oriented and orientation of system are towards component based system. These results points out that most of the object oriented systems are becomes legacy now and software developers are more interested in migrating object oriented systems into component based systems. The main drivers of reengineering is changing technology and high maintenance cost and lack of expertise, adaptability and lack of documentation are the major challenge faced during reengineering process. Results also shows that reengineered legacy systems are having proven technology over original legacy system.

The paper presented the state of art in the reengineering and revolution of legacy systems. The need of reengineering of object oriented legacy system and some potential benefits of component based system are also outlined in this paper. Although this review has explored the field of reengineering, further research is needed to confirm the obtained results. Future research includes the inclusion of more resources form conferences or workshops and exploration of the other domains and in future, the findings will be validated by using some real world legacy reengineered system. It may be possible that

the several important advances is already taken place but there are still many areas for future research that will benefit the software communities in every aspect.

References

[1] A.-P. Li, Z.-h. Wang, L.-G. Duan and X.–P. Li, "Study and application of legacy system reengineering based on component reuse", Journal of Applied Sciences, vol. 13, no. 8, **(2013)**, pp. 1233-1238.
[2] A.-E. El Hamdouni1, A. Djamel Seriai1 and M. Huchard, "Component-based Architecture Recovery from Object Oriented Systems via Relational Concept Analysis", LIRMM/CNRS, Montpellier 2 University, France, **(2010)**.
[3] A. Hessellund, "Refactoring as a Technique for the Reengineering of Legacy Systems", ITU, Kobenhavn, **(2004)** April.
[4] Anne-Marie and Radu, "Metrics-Based Problem Detection in Object-Oriented Legacy Systems Using Audit-Reengineer", Software Engineering Division - Sema Group Sae, Spain, **(2000)**.
[5] B. Arie, "From Legacy to Component: Software Renovation in Three Steps", CAP Gemini, (2000).
[6] A. Jatain Arpita and Dr. D. Gaur, "Review based on Data Clustering Algorithms", IEEE International Conference on Information and Communication Technologies, 978-1-4673-5758-6, **(2013)**.
[7] A. Sharma, "A Critical Survey of Reusability Aspects for Component-Based Systems", World Academy of Science, Engineering and Technology, **(2007)**.
[8] A. P. Sinha and H. Jain, "Ease of reuse: An empirical comparison of components and objects", IEEE Software, published by the IEEE Computer society, **(2013)**.
[9] B. Caprile, A. Potrich and P. Tonella, "Reverse Engineering of Object Oriented Code", Geneva (CH), **(1999)**.
[10] C. Szyperski, "Component Software; Beyond Object-Oriented Programming", Addison-Wesley, **(1998)**.
[11] D. A. Jackson, K. M. Somers and H. H. Harvey, "Similarity coefficients: measures of co occurrence and association or simply measures of occurrence", The American Naturalist, **(1989)**, pp. 436–453.
[12] D. Staffan and D. Sandell, "Reengineering and Reengineering Patterns", The Department for Computer Science and Engineering, Mälardalens Högskola 2002-02-24 Västerås.
[13] D. Ganesan and Jens Knodel, "Identifying Domain-Specific Reusable Components from Existing OO Systems to Support Product line Migration, EUREKA 2023/ITEA-ip00009, **(2005)**.
[14] L. H. Rosenberg, "Software Re-engineering", Lawrence E. Hyatt Manager, Software Assurance Technology Center System Reliability and Safety Office Goddard Space Flight Center, 1997, NASA 301-286-7475.
[15] E. Holz and O. Kath, "Manufacturing Software Components from Object-Oriented Design Models", IEEE Xplore, **(2001)**, pp. 262-272.
[16] E. Marchetti, F. Martelli and A. Polini, "Refactoring a Legacy system using component", www.iei.pi.cnr.it/ERI, **(2004)**.
[17] E. J. Chikofsky and J. H. Cross II, "Reverse engineering and design recovery: A taxonomy", Index Technology Corp. and Northeastern University and Auburn university, **(1990)**.
[18] E. Lee, B. Lee, W. Shin and C. 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), IEEE, **(2003)**.
[19] F. C. Meng, D. C. Zhan and X. F. Xu, "Business component identification of enterprise information system: a hierarchical clustering method", Proceedings of the IEEE International Conference on e-Business Engineering, **(2005)** October, pp. 473–480.
[20] W. B. Frakes and K. C. Kang, "Software Reuse Research: Status and Future", IEEE Transactions on Software Engineering, vol. 31, no. 07, **(2005)**, pp. 529-536.
[21] G. Abowd, A. Goel, D. E Jerding, M. McCracken, M. Moore, J. William Murdock, C. Potts, S. Rugaber and L. Wills, "MORALE: Mission Oriented Architectural Legacy Evolution", IEEE Explore, **(1997)**.
[22] H. Jain, "Business Component Identification - A Formal Approach", IEEE Xplore, **(2001)**, pp. 183-187.
[23] H. S. Hamza, "A Framework for Identifying Reusable Software Components Using Formal Concept Analysis", Sixth International Conference on Information Technology: New Generations, IEEE, **(2009)**.
[24] H. Li, H. Guo, H. Guan, X. Feng, Y. Xu and H. Yang, "An evolution scheme for business rule based legacy systems", Journal of Theoretical and Applied Information Technology, ISSN: 1992-8645, vol. 47, no. 1, **(2013)** January.
[25] H. Mathkour, A. Touir, H. Hakami and G. Assassa, "On the transformation of Object Oriented-based Systems to Component-based Systems", IEEE International Conference on Signal Image Technology and Internet Based Systems, **(2008)**.
[26] H. Washizakia and Y. Fukazawab, "A technique for automatic component extraction from object-oriented programs by refactoring", Science of computer programming, Elsevier, **(2005)**, pp. 99-116.
[27] H. Washizaki, H. Yamamoto and Y. Fukazawa, "A Metrics Suite for Measuring Reusability of Software Components", Software Metrics Symposium (METRICS'03), IEEE, **(2003)**.
[28] H. K. Kim and Y. K. Chung, "Transforming a Legacy System into Components", Springer-Verlag Berlin Heidelberg, **(2006)**.

[29] Holger, "The FAMOOS Object-Oriented Reengineering Handbook", ESPRIT program Project no. 21975 (FAMOOS), Swiss Government under Project no.NFS-2000-46947.96 and BBW-96.0015, **(1999)** October.
[30] Z. Hui, B. Kerong and W. Hongbo, "Modeling and implementing of service –oriented reengineering process of legacy system", Journal of Naval University Engineering, **(2010)**, pp. 24-30.
[31] I. Sommerville, Software Reengineering, book, **(2000)**.
[32] J. Feng Cui and H. Seok Chae, "Applying agglomerative hierarchical clustering algorithms to component identification for legacy systems", Journal of Information and Software Technology, Elsevier, **(2011)**, pp. 601–614.
[33] S. Jong, W. Soo and Dong, "Component Identification Method with Coupling and Cohesion", Proceedings of the Eighth Asia-Pacific Software Engineering Conference, 1530-1362/01 2001 IEEE.
[34] K. Bergner and A. Rausch, "Putting the Parts Together-Concepts, Description Techniques, and Development Process for Component ware", Proceedings of the Thirty-Third Annual Hawaii International Conference on System Sciences, vol. 8, **(2000)**.
[35] K. K. Aggarwal, Y. Singh, A. Kaur and R. Malhotra, "Software Reuse Metrics for Object-Oriented Systems", Software Engineering Research, Management and Applications (SERA'05) IEEE, **(2005)**.
[36] M. Bauer and M. Trifu, "Architecture-aware adaptive clustering of OO systems", Proceedings of the 8th European Conference on Software Maintenance and Reengineering (CSMR '04), Tampere, Finland, **(2004)** March, pp. 3–14.
[37] M. Shtern and V. Tzerpos, "Clustering Methodologies for Software Engineering", Hindawi Publishing Corporation, Advances in Software Engineering, **(2012)**.
[38] M. Lanza, "Object-Oriented Reverse Engineering Coarse-grained", Fine-grained, and Evolutionary Software Visualization, Leiter der Arbeit: Prof. Dr. S. Ducasse, Prof. Dr. O. Nierstrasz Institut f ¨ ur Informatik und angewandte Mathematik, **(2003)**.
[39] M. A. Serrano, D. L. Carver and C. Montes de Oca, "Reengineering legacy systems for distributed environments", Journal of System Software, Elsevier, vol. 64, **(2002)**, pp. 37–55, **(2002)**.
[40] N. Sing Gill, "Reusability Issues in Component-Based Development", ACM Sig Soft Notes, **(2002)**.
[41] A. O'Donell, "Building Blocks of Success", Insurance and Technology, **(2001)**, pp 41-5.
[42] O. Maqbool and H. A. Babri, "Hierarchical clustering for software architecture recover", IEEE Transactions on Software Engineering, vol. 33, no. 11, **(2007)**, pp. 759–780.
[43] O. Maqbool and H. A. Babri, "The weighted combined algorithm: a linkage algorithm for software clustering", Proceedings of Software Maintenance and Reengineering, **(2004)** March, pp. 15–24.
[44] S. Oscar, "CDIF as the interchange format between reengineering tools", NFS-2000-46947.96 and BBW- 96.0015, **(1999)**.
[45] P. Dugerdil, "Using RUP to reverse-engineer a legacy system", IBM Journal, **(2006)** September.
[46] P. Newcomb and R. A. Doblar, "Automated Transformation of legacy system", www.stsc.hill.at.mil, **(2001)** December.
[47] S. Demeyer, S. Ducasse, R. Nebbe, O. Nierstrasz and T. Richner, "Using Restructuring Transformations to Reengineer Object-Oriented Systems, Software Composition Group, University of Berne, **(2000)**.
[48] S. Budhkar and A. Gopal, "Component identification from existing object oriented system using Hierarchical clustering", IOSR Journal of Engineering, vol. 2, no. 5, **(2012)** May, pp. 1064-1068.
[49] S. Budhakar and A. Gopal, "Component based architecture recovery from object oriented systems using existing dependencies among classes", International journal of computational Intelligence Techniques, ISSN: 0976-0466, vol. 3, no. 1, **(2012)**, pp. 56-59.
[50] S. Houari Simon and Regis, "From object oriented applications to component Oriented Applications via component oriented Architecture", Published in 9th working IEEE/FIP conference on Software Architecture, Boulder, USA, **(2011)**.
[51] S. K. Mishra, D. S. Kushwaha and A. K. Misra, "Creating Reusable Software Component vfrom Object-Oriented Legacy System through Reverse Engineering", Journal of Object Technology Published by ETH Zurich, Chair of Software Engineering, vol. 8, no. 5, **(2009)** July-August.
[52] P. Sudhakar and P. Sakthivel, "Reengineering Legacy to Modern System with One Time Checker for Information System Evolution", American Journal of Applied Sciences, 832-841, ISSN 1546-9239, **(2012)**.
[53] S. Chardigny, A. Seriai, M. Oussalah and D. Tamzalit, "Extraction of Component-Based Architecture From Object-Oriented Systems", Seventh Working IEEE/IFIP Conference on Software Architecture, IEEE, **(2008)**.
[54] C. Szyperski, "Component Software-Beyond Object-Oriented Programming", Second Edition. Addision-Wesley, **(2002)**.
[55] T. A. Wiggerts, "Using clustering algorithms in legacy systems remodularization", Proceedings of the 4th Working Conference on Reverse Engineering, **(1997)** October, pp. 33–43.

