CHAPTER - IV
EXISTING RE-ENGINEERING MODEL(S)

4.1 Introduction

Study of the existing model(s) is pre-request for designing the re-engineering model(s) for object-oriented software systems. Re-engineering of software systems is very hot area and in coming future, it will also gain more importance when software will exist everywhere. It is potential and promising area in the software industry. Re-engineering is growing but not at fast rate as it is required and software backlog is increasing. Software developers and engineers are developing Software as jobs and processes are going to be automated everywhere. In this age of information technology, software is working in every sphere of life and is covering more and more areas. But the perception of re-engineering is slow. When business process is changed or requirements are changed the software managers go for new development and the old legacy software is wasted and is added to the backlog. With this sorry state of affairs the software investments graph go high which is difficult for the organizations. Promotion of re-engineering of software systems is the solutions to this situation. Re-engineering model(s), tools and techniques should be discovered to make the re-engineering practice common.

There is confusion in the terminology of re-engineering. Every system may be abstract system or some hard ware system can be re-engineered for high performance, quality service and to increase life span of the system. Here re-engineering means re-engineering of software systems only. Re-engineering is more applicable to software systems as compared to other type of systems because software systems do not wear or tear out. There is no depreciation in case of software systems. Only change in our demands, requirements matters in software systems.

The aim of the study is to propose re-engineering model(s) for re-engineering object-oriented software systems. Re-engineering is more relevant for object–orient software as object-oriented systems are more reusable. Systems of this type are easy
to understand and the complexity is resolved by separating objects. In procedural oriented programming it was difficult to resolve complexity as large program is a concrete bundle of millions of instructions. The idea of objects came from Nature, as Nature has created object-oriented systems. Maintenance and re-engineering is easy in case of object-oriented software systems.

Model(s), tools and techniques are the requirement to make the re-engineering widespread. Reengineering cost, time and risk will also be reduced with the tools. Many models of re-engineering for software systems exist and are applied for re-engineering. These models were thoroughly studied, their limitations, potentials their presuppositions and consequences. With the change in software developing technology, model(s) become old and less applicable to re-engineering process. Existing Model(s) are not accepted universally and also there are many re-engineering activities for the models do not exist. There exist gaps in performing the complete process. Existing model(s) of re-engineering software systems in the software industry are mostly applicable to procedural oriented software systems. The software industries mostly shifted to object-oriented technology for software development. These model(s) are very helpful to software engineers and developers for re-engineering software systems. There is no doubt present state of the practice needs defined models and software tools. Following are the existing models for re-engineering software systems from time to time.

4.2 Horse Shoe Model

This model was proposed by Bergey et al. [9], Carnegie Mellon Software Engineering Institute CMU/SEI, TN, 2000. This is the most basic model of re-engineering. In this model there are three logical steps- analysis of the existing system, transformation of the architecture, and integration of the new system. First step is reverse engineering- go up, second step- logical transformation of the architecture-go right and then redevelopment (forward engineering)-go down which in all form the basis of horseshoe. It is as shown below:
Figure 4.1 Horse Shoe Model

Source: [http://www.sei.cmu.edu](http://www.sei.cmu.edu)

Software Engineering Institute (SEI), Carnegie Mellon University, Pittsburgh.

In this model software engineers go from code level to higher abstraction levels. In this step architecture is recovered by generating artifacts from the source code and we entered in the second stage. Something is added or something is deleted according to new requirements and architecture is redesigned according to new technology and we are ready to enter in the next stage and that is the 3rd and the last stage in horseshoe model. In this stage new architecture based development is done.

This is complete model for re-engineering software systems. The three basic re-engineering processes are as they fit into the horseshoe model. In its most fundamental form there are three basic re-engineering processes

1. Analysis of existing system
2. Logical transformation
3. Development of a new system

This model is useful for software engineers to re-engineering legacy software systems. It describes three level of re-engineering. It is simple and quicker to apply for re-engineering.
Limitations:

- Re-engineering cost and other factors are ignored
- It is costly as it is for re-engineering the whole system
- Hard to add new functionality
- It ignores the reusability of software modules
- It is suitable for procedural oriented software systems
- Stand-alone re-engineering tool, no interoperability
- It does not specify re-engineering approach

4.3 CORUM Model II

Program understanding software tools are presently not interoperable, researchers and engineers waste time and efforts in reinventing already existing tools. Commercial organizations that have been setup to support the construction of program understanding tools have serious flaws in this regard. Woods et al. [128] proposed CORUM (Common Object-based Re-engineering Unified Model) framework which could represent information about Code-Base Management System (CBMS). Its architecture is to support interoperability between program understanding tools. It was designed to provide a mechanism for storing extracted information and products (like Symbol Tables, CFG, DFGs, data slices and programming plans). Code level re-engineering tools are for creations and manipulation of Abstract Syntax Trees (ASTs), Control Flow Graphs (CFGs), and Data Flow Graphs (DFGs). It also provided inter-operability between tools that accessed and produced information. Commercial CBMS has four parts (1) language gateways, (2) an information model, (3) processing tools, and (4) a component repository. Language gateways were made of tools that were capable of parsing and loading information about source code into the CORUM representations. CORUM initial was information model. The information was generated directly from source code. There are a large number of stand-alone re-engineering tools that work at different levels of abstraction ranging from code-level to software architecture. Re-engineering tools are needed at each processing level and at the same time they
should be compatible and they should be sharing the information. Extended framework—called CORUM II was built up to overcome this problem. All the tools were integrated as to do re-engineering complete system at all the levels. CORUM II—model is organized around the metaphor of a “horseshoe”, where left hand side consists of facts extractions, right hand side consists of development activities, and the bridge between the sides is for transformations of architecture from old to new.

Architectural re-engineering tools have several characteristics in common with the code-level tools and at the same time they have some significant differences. Architecture tools usually use a function-based representation rather than Abstract Syntax Trees based representation of code.

Figure 4.2 CORUM II

Source:  http://www.cs.kent.edu
Software Engineering Institute, Carnegie Mellon University, Pittsburgh

CORUM II the extended framework is a part of larger context at Software Engineering Institute (SEI) which is sufficient to support both re-engineering analysis and architectural reasoning, and to the integration and interoperability of re-engineering tools within the context of a Code-Base Management System (CBMS).

The application of this model is integration and interoperability of re-engineering tools. This model is community wide efforts at a common platform for re-engineering. It is a framework for integration of architectural and code based re-engineering tools.

**LIMITATIONS**

- It focuses on interoperation of tools only.
- In itself does not give any technique of re-engineering
- Cost and risks and other factors are ignored.
- It ignores the reusability of software modules
- It does not specify re-engineering approach

**4.4 Byrne Model**

E. J. Byrne [18] has proposed a process model for re-engineering with three parts: Reverse Engineering, Alteration, Forward engineering which he visualized as under
This model in the above Figure 4.3 applies three principles of re-engineering: abstraction, alteration and refinement. Refinement is done at the four levels in the process. Abstraction levels are created and generated by reverse engineering of old software systems. Alteration has two dimensions: 1) change in functionality 2) change in implementation. To change characteristics of the old software, changes are done at the abstraction levels. With the increase in the abstraction levels, type of the tasks is changed and also reverse engineering level are increased. If change is required at the code level that is we want to change the programming language of the code, it can be done at the implementation level. In this type of change reverse engineering is not needed. The authors discusses three reengineering approaches for the model given below.
• Big Bang approach
• Incremental/Phase-out approach
• Evolutionary approach

In this model any one of the three approaches can be applied. It depends upon the type of the task and level of abstractions. Horse-shoe model aims to integrate the code level and the architecture level re-engineering views. This model for software re-engineering shows the processes at all levels of re-engineering based on the levels of abstraction used in software development. The upward ride is termed reverse engineering and has associated sub-processes, tools and techniques at all four levels of software development. Alteration between two sides is to make one or more changes to a system by including addition, deletion and modification of information. Abstraction level remains the same at any particular level according to the task. Successive replacement of existing system information with more detailed information is done on right hand side. It is gradual increase in refinement and decrease in abstraction levels. This is called forward engineering. At the lowest level we reach with developed new code.

In its most fundamental form, there are also three basic re-engineering processes.

(i) Architecture recovery

(ii) Architecture transformation

(iii) Architecture-based Development.

This model is very useful for re-engineering software systems. It is flexible for changes at various levels of reengineering. Software engineers can use this model for reengineering the software systems at any of the four levels depicted in the figure 4.3. It specifies four layers in all the three re-engineering processes. At every step in each layer there can be some tool or technique designed to do that very step.
Limitations

- It is not useful for software managers as it does not estimate re-engineering cost and right time for re-engineering.
- Risk and other factors involved in re-engineering are ignored.
- It is costly as it also recommends re-engineering process for the whole system
- It is also suitable for procedural oriented software systems
- Stand-alone re-engineering tool, no interoperability

4.5 Dual-Spiral Re-engineering Model

The Dual-Spiral Re-engineering Model is given by Yang et al [132] is depicted in the following Figure 4.4. The main workflow in Dual-Spiral Re-engineering Model requires the two systems one is legacy and other is target system to work together. A communication proxy system is added between the two systems to facilitate them working as a complete system in the whole process. In this model functionality is moved from candidate system to target system step by step as in spiral model for software development. During the entire process, the active functionality in the candidate legacy system is in a decremental pattern, and the active functionality in the new reengineered system is in an incremental pattern, as in Fig. 4.4. This process also facilitates the addition of new features while transitioning the existing functionality from candidate to target system. The implementation of the proxy can vary based on the specifics of the architectures involved, e.g. scripts, database triggers, etc. The re-engineering process in Dual-Spiral Model is divided into three main steps: divide the functionality, start the spiral procedure, and terminate the spiral procedure and continue this loop till we get a new reengineered software system.

This model is useful for re-engineering as there is less risk involved in it. It is good model as gradual transition is done form candidate system to target system. There are no chances of failure of the re-engineering projects with this technique. Reengineering can be achieved during working so the work does not suffer.
Fig. 4.4 Dual-spiral Re-engineering Model By Xiaohu Yang et al.
Source: Proc. TENCON 2005 IEEE Region 10 conference
Limitations

- It needs a proxy system for re-engineering process.
- It does not specify the criteria for dividing the functionality.
- Reengineering is not achieved by moving the functionality from one system to another.
- It ignores the big bang reengineering approach.

4.6 Rainfall Model

Rainfall Model for re-engineering was proposed by Singh et al. [99] is shown in the following figure 4.5. In this model the authors do not re-engineer the whole system as done in all other models discussed above. The part of the software system is identified for re-engineering. The authors call this part as candidate part for re-engineering and this is the first step in this model. This model has five processes.

The authors call this model as ‘Rainfall Model’ by relating this system of re-engineering with rainfall system. In rainfall system first some water (some part of software- candidate part) evaporates above and make clouds. The clouds merge with other clouds and their shape may change (architecture transformation). Then it rains (forward engineering). The rained water then integrates with water already present (integration). In this model, process of re-engineering is divided into five steps as shown below:

1. Identifying the candidate part of the system
2. Reverse engineering of that part
3. Architecture transformation of that part
4. Forward engineering
5. Integration

If the whole system is to be reengineered then first and the last steps are omitted.
Figure 4.5 Rain fall Model


The last stage in this model is integration. After re-engineering the identified part of the software, it is integrated with the original system.

This model is economical and better than other models as it do not recommend reengineering the whole system. The first step is to identify the portion to be reengineered, But the author did not give any technique to identify and separate the faulty (candidate part) portion for reengineering.

This model is based on the natural automatic re-engineering process. Automatically hot and dirty water goes on evaporating and accumulated in the form of clouds in the sky. Then after time with the change in the weather, it rains pure water to the ground and mixed into the original water to make the water table full and clean. Dirty water is compared with the faulty portion of the software which is to be reengineered like pure water by nature.
**Problems with this model**

The first step is to identify the part of the software to be reengineered. But it is not explained how to identify that part? The software part which is to be reengineered is called candidate part by the authors. The authors did not suggest any method or technique to identify the candidate part. On what ground candidate part will be stripped. If the whole system is candidate part then the scenario is changed. If some part is identified and reengineered, then the problem of plugging in the original software is not addressed. But if these issues are addressed then it is fine model to reengineer the software system. In this model whole software is rarely to be reengineered, and only the candidate part is re-engineered.

This model of re-engineering is automated in nature and good for re-engineering software system. Cost of re-engineering is controlled as the whole software is not re-engineered in this technique. The risk factor is also less as the re-engineering is done fragment wise and the whole software is not touched. This is economical method.

Still this model has the following limitations:

- It is difficult to identify and separate the candidate part
- No provision for additional functionality
- Stand-alone re-engineering technique, no interoperability
- This method is fully automated (hundred percent automation) which is difficult to achieve.

**4.7 Reflexion Model**

The reflexion model enables a software engineer or developer to summarize a large body of source code in terms of a selected, high-level, structural view of the system. Re-engineering software system needs understanding of system’s source code. Murphy [77] developed the software reflexion model technique for gaining insight into the source code. Microsoft engineers used this technique as an experimental for re-engineering Microsoft Excel code (1.2 billions lines of code). It is a technique to aid software engineers in understanding and reasoning about the
structure of large software systems’ source code. What is structure of software system? Structure of software system can be defined as a set of interconnecting parts of any system. Reflexion model allows a software engineers to view the structure of a system's large source code through a chosen high-level architectural view. It is an approach that helps software engineers to use a high-level model of the structure of an existing software system as a lens through which to see a model of that system's source code. An engineer defines a high-level model and specifies how the model maps to the source code. A software tool then computes a software reflexion model that shows where the engineer's high-level model agrees with and where it differs from a model of the source code. The name of the software tool used is RMTool.

Figure 4.6 Reflexion Model Techniques

Source: [http://www.cs.ubc.ca](http://www.cs.ubc.ca)  
Ph.D. Thesis, Gail C. Murphy, University of Washington, 1996

This model is very useful to understand the architectural view of large source program. The RMTTool generates the high level architectural view of the source code which is easy to understand by the software engineers. The circled numbers refer to
the steps in the process of generating the structural view of the system’s source code. Briefly in the Software reflexion model technique the engineers define (1) high-level model, (2) extract a source model, (3) define a map, (4) compute a reflexion model and then (5) investigate and refine the reflexion model of software source code. Upon investigation of the reflexion model, the user may iteratively refine the inputs and recompute a new reflexion model. Derive a software reflexion model and iteratively go on refining it

This is used for understanding the large source code which is needed for software maintenance and re-engineering. Software tool is used to generate architectural view from the source code. It is useful for maintenance as well as for re-engineering.

Limitations:

- It is only used for understanding of source code which is not a solution to complete re-engineering process.
- Level of refining software reflexion generated with this method is not defined. As there is no bunch mark to stop, processing will be in endless loop.
- Some time there is no need to understand the whole system’s source code
- It is partial re-engineering model

4.8 Service-Oriented Software Reengineering (SoSR) Model

Source: Proceeding, IEEE 40th Annual Hawaii International Conference on System Sciences (HICSS’07)

Sam Chung et al. Proposed service-oriented software re-engineering model for modernization of tightly coupled aged legacy information systems to the loosely coupled, agile, and service-oriented information systems. Loosely coupled software components can be integrated to other software systems by SOC (Service-Oriented Computing). SOC provides standardized service interfaces and interaction protocols
to plug the software as services to the legacy systems. In SOC environment system
is more naturally integrated as compared to the current distributed middleware
technologies such as RMI, CORBA, and mobile agents.

The purpose of this model is to propose a Service-Oriented Software
Reengineering (SoSR) technique/method that is applied to a legacy software system
for reengineering it into a service-oriented software system by using SOC. In order
to apply this SoSR methodology, a Service-Oriented Architecture (SOA) is needed
which is a model of three-service-participants, Service Consumer, Service Broker,
and Service Provider. Re-engineering of a legacy system is defined in terms the
tasks of the three participants. SOA is an architectural model in which a service is
published to a service registry and the published service is discovered and bound to
a service client when the client requests for the service. If the web services
technologies are used an interface of the software component is represented in
WSDL (Web Services Description Language).

In SoSR model reverse software engineering and service-oriented forward
software engineering, is done. In the reengineering process of the legacy system
requirements are updated according to user’s new requirements. This new set of
requirements describe what features of a legacy system will be modernized to get the
target system with SOC service.

The SoSR methodology consists of two main processes. One is the reverse
engineering of the software and the other is service-oriented forward engineering of
the software system. Through the reverse engineering of software process, a visual
model for the legacy system is constructed by analyzing the given legacy system and
the modernization requirements. Based upon this model for the legacy system and
the modernization requirements, a target system is generated through the service-
oriented forward software engineering process.
Fig. 4.7 SoSR Model

Source: http://ieeexplore.ieee.org

Several concepts are employed to achieve service-oriented software reengineering and that are 3-layered architecture, n-tier architecture, SOA (Service-Oriented Architecture), SOC (Service-Oriented Computing), business process design and execution, role-base model, 4+1 view model, RACI chart. In the reverse engineering process, a visual model for the legacy system is constructed by analyzing the candidate (legacy) legacy system and modernizing the requirements. On the basis of current requirements target system is generated through a service-oriented forward software engineering process. First it is examined how the three-service-participants model is related to the 3-layered architecture. Based upon this model reengineering of the legacy system is defined in terms of other view model (4+1 View Model). The 4+1 view for each participant is further developed in terms of tasks and different roles by using RACI (Responsible, Accountable, Consulted, and Keep Informed) charts. Then reverse engineering process is conducted based on the RACI chart and legacy system is documented as visual model. The visual model for the legacy software is used for the next process that is service-oriented forward software engineering and the target system is achieved.
The legacy aged system is migrated to a distributed environment and can be wrapped and exposed as a web service.

Legacy system can be embedded with reliable functionality of valuable business logic.

The embedded functionalities can be used as independent services.

**Limitations**

This strategy does not change the fundamental characteristics of the legacy system that are being integrated and wrapped.

It can not solve the problems already present in the legacy software, such as problems of maintenance and improvement.

**4.9 Parallel Iterative Re-engineering Model**

**Source:** Proceedings of the 2009 IEEE International Conference on Systems, Man, and Cybernetics San Antonio, TX, USA - October 2009

This model was proposed by Xing Su et al. [131] for reducing the re-engineering process time. From time to time improvements have been made in the re-engineering methodologies. It allows various components and data to be operated in one cycle during the re-engineering process. Its main idea is to analyze the concepts in a relationship concept lattice applying “Bottom-Up” algorithm. Based on the concept lattice obtained in last scenario, developers can already draw the schedule plan according to project’s resources. However the workloads can be very heavy providing a complex lattice. Hence the relationships in concept lattice need further process. ‘Bottom-Up’ algorithm is designed to meet this requirement. The algorithm consists of two parts: the Concept Partition Algorithm and the Recursive Sorting Algorithm. An Iterative Cycle Reference Table (ICRT) is delivered at the end to help schedule the iterative cycles. The authors gave four steps in total to fulfill the parallel iterative re-engineering model. ICRT contains all optimized information of the access relationships in the legacy system. Each role within it covers the essence information of a node in the concept lattice, including elements of priority, name of the access concept, the legacy components and data that need re-engineering.
Parallel iterative re-engineering model proposed by Xing Su et al. uses rapid re-engineering process and uses fully the developed resources. It allows several components and shared data to be migrated in one cycle according to their access relationships. Also, the Iterative Cycle Reference Table built by ‘Bottom-up’ algorithm helps developers to allocate developing resources according to the re-engineering schedule, which can make a full use of resources in some extent.

To gain a more effective solution, threshold of access strength would be one of the main focuses in building the access relationship table. Furthermore, the authors suggest for focusing on the system structure and seeking for solutions to enhance the performance of transition system in the re-engineering process. The applications of the model are as given under:

- It is a rapid re-engineering process and less time is required for re-engineering
- It allows several components and shared data to be migrated in one cycle.
- Its algorithm helps software engineers to allocate developing resources according to the re-engineering schedule

**Limitations**

- It is to reduce the reengineering process duration but unfortunately duration of reengineering process is still quite long.
- Workload can be very heavy with complex lattice which further needs processing

Software re-engineering model(s) discussed above all are stand alone models. In all the models, the first step is to examine the source code and extract the design and the artifacts. Accommodate the new requirements, new business process. Then redesign the structure with new technology for the new platform. Then coding is done according to new design. There are many issues which remain to be unsolved in the above existing models. A model is a guide map to complete a re-engineering process for some software system. If some stage from beginning to end is uncovered, remains unexplained then it will be trouble for the people who are responsible for the assignment of re-engineering software system. In this way there were gaps in the re-engineering process solutions.
Software re-engineering models are used by software engineers and managers to reengineer the legacy application software systems. Up to date existing re-engineering models were examined for re-engineering software systems. It was found that there are short falls at many places in the whole re-engineering process. Re-engineering process has main three stages and each stage has many activities to perform on the legacy software systems. There are many sub processes for which re-engineering models even do not exits. These gaps in the re-engineering process are trouble for software managers as well as for software engineers. They have faith in software maintenance and/or new development. The short falls detected in applications of the existing models were tried to remove in the proposed models.

Software managers face the problem of knowing the right time for re-engineering the software. If the software is reengineered at the right time, the cost of re-engineering is minimum cost. There is no existing re-engineering model which specify the right time for re-engineering. Software managers go on maintaining the software even with high maintenance cost which further deteriorates the architecture of the software system and ultimately legacy software goes to the backlog of software. Software managers find little difference between software maintenance and re-engineering. Both these terms are used interchangeability in the software industry and costly software is added to the software backlog due to this confusion.

Most of the existing models recommend for re-engineering the whole software, and it adds to the cost/efforts of re-engineering. Only a small part of the software can creates problems, and if only that part is reengineered it will take less cost, time and less risk will be involved.

The cost of re-engineering is difficult to measure, which is big problem in deciding for re-engineering of software. Without knowing the cost it is not easy for both vendors and clients to negotiate for re-engineering project. Risk factor is also involved in it, how much risk is there? How much time it will take? There are no such models which provide the solutions of these questions.

With the software tools and techniques for re-engineering, step wise automation can be achieved. With the precise re-engineering models and methods integrated at the common platform, the re-engineering practice can be made widespread. In this way, software investments of the organizations will significantly fall.