CHAPTER - I
INTRODUCTION

Re-engineering is a set of activities that are carried out to change old legacy software system to reconstitute it as a new software system. Re-engineering is done to enhance performance, reduce maintenance, and to cop up with changed requirements and environment. The concept of re-engineering originated from the hardware world in the nineteenth century. A system (natural or man made) originates and expires after a certain time according to the inevitable law of nature. Everything undergoes formation and growth on one hand and decay on the other. Every system on the earth has a certain age (life span). There are many types of systems and each type of system has its own features. The age concept is different for software systems. A software system works forever as it do not wear and tear out. But when it is old? It is old when our business process changes and/or it does not meet our requirements and/or does not conform to environment in which it works.

1.1 System

The word ‘System’ is embedded in the title of the thesis. It is often used in our daily life. System theory is a vast subject of importance. Every thing on the earth works in the form of a system. Examples of the systems are educational system, political system, solar system, computer system, software system and so on. Systems are of many types, each system falls in some or the other category. System may be defined as a component or group of interrelated components working together (as a whole), accepting input and producing output in a definite time period. Input is transformed into output based on certain transformation process.

Fig. 1.1
Components of the system are input, process, output, feedback, control, environment, boundary and interface. System always exits in some environment.

System is a set of interacting or interdependent entities real or abstract integrated to work together. It has definite input and output. Object which is not a system is unorganized material of no use. It has no input and no output.

The word ‘Model(s)’ is embedded in the title of the thesis ‘Design of model(s) re-engineering object oriented software systems’. It will be worth explaining the word ‘model(s)’. The word ‘Model(s)’ refers to model or models. It means designing one or more models. Model is a graphical, mathematical, symbolic, physical, or verbal or conceptual representation or simplified description of a concept, phenomenon, event, relationship, configuration, structure, system, or an aspect on the planet. Modeling helps in understanding, decision making, controlling, predicting and improving the system. Modeling is very helpful and even necessary where systems are large and complex to understand and control. Design of model(s) is necessary to refine and improve the system or working of the system.

Software System

A program or group of programs designed for the end users to accomplish a particular task is called a software system. This system is man made and is closed form system. Software system takes input, processes it and produces useful information. When our requirements, business process, technology and/or environment changes software system does lose support. Then we retire (throw the software) the software. Developing software involves many activities in addition to writing code. The IEEE, in their standard 610.12-1990, defines software as ‘computer programs, procedures and possibly associated documentation and data pertaining to the operations of a computer system’. Kishore [64] defines computer program as a set of instructions to perform some specific task by the computer. The number of instructions in a program depends upon the size and complexity of the task. If our problem consists of a number of complex tasks, we require many computer programs to complete our task. The collection of computer programs or procedures to perform that task on the computer is called software system or simply software.
**Object-Oriented Software System**

The idea of object-oriented software systems came from object oriented physical systems. The Mighty Nature has created object oriented systems as better service and maintenance is possible. Now software engineers designed object oriented software systems on the pattern of physical systems. A software system is a collaboration of objects to work together. Object oriented system can be understood easily as each object embodies an amount of complexity that programmers can fully grasp. Object oriented technology uses objects and not algorithms as its fundamental building blocks. In this context, object is a conceptual module that can be plugged in and plugged out from the software system.

**What is an Object?**

Object is the basic component of design. It is abstraction of module of the software that can be independently designed, tested and plugged into the rest of the software. It is a bundle of data and instructions interfaced with the software system. The idea of object is not for computer machines but it is useful for software engineers to design quality software with straightforwardness. Object can be big or small it depends upon the number of attributes of the object used in the programming. The concept of Object is equally important in designing as well as in re-engineering the software system.

![Diagram of an Object](image)

**Fig. 1.2** an Object

An object is locked box which contains instructions and data. Stefik and Bobrow define objects as “Entities that combine the properties of procedures and data since they perform computations and save local state” Stefik [106]. Objects are bundles of data and operations kept private from the rest of the software system. In
procedural oriented programming data and code are kept separate. Objects receive and send messages. We need not know the details of the objects, we access only through the messages. Keeping every thing private about object is called information hiding. Objects oriented languages are good for reuse. Inheritance promotes reuse as we do not start from scratch when to write new program. Inheritance depth increases the reusability of the software. Degree of coupling of objects also matters. Higher degree of coupling cause difficulties in software maintenance as it increases complexity. Independent objects are more reusable.

![Diagram](image)

**Fig. 1.3** Eight objects software system

Limitation of human capacity for dealing with complexity is the need to break up the system to understand and deal with it. Object oriented system can be understood easily at higher level of abstractions, and is viewed as different objects. In successive generation of languages decomposition is supported differently. The advancement in the software technology, from machine language to assembly to procedural to object-oriented languages, helps programmers to make their jobs easier accordingly. In this context, object is like the real world objects, a conceptual module than can be plugged in and plugged out from the software system.

1.2 Software Re-engineering

Before discussing software re-engineering, it will be use full to discuss software engineering as software re-engineering is a branch of software engineering. Also software re-engineering came later in the software industries.

Software engineering

When first digital computers appeared, hardware aspect of the technology considered significant. Engineers were doing efforts to design more powerful
computers. Hardware development were considered to be significant, software was of little importance. Customers and end-users could write small computer programs. Software in those days was not so importance, only small number-crunching programs were written by the users of the computer. With the passage of time hardware processing capacity increased vastly. End-users and customers were not able to write programs for complex problems and failed to exploit available hardware technology. This gap increased to a great extent and software crises felt. It was felt that software development is not just writing a small code but involves many activities. Many software projects failed and hence main focus was on the software development. Software development was high technical and was thought of product of engineering like other subjects that require high technical skills. The term ‘software engineering’ was coined in 1967 by Friedrich Bauer at a pre-conference meeting in Germany at a NATO (North Atlantic Treaty Organization) conference on issues in developing large scale software systems Thayer [111]. IEEE Standard 610.12-1990 has defined software engineering as “the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is the application of engineering to software” Kishore [63]. In software development we go through the processes like analysis, design, coding, testing, documentation and maintenance of the software. We apply engineering science to all these processes and therefore software engineering came into existence.

Re-engineering

Software re-engineering is relatively new sub discipline within software engineering. Re-engineering has many synonyms terms like renovation, refurbishing, reclamation, overhauling. According to Arnold (1994) there are no universally accepted spellings of re-engineering. The most common spellings are re-engineering and reengineering. Re-engineering can be done of any type of system which is aged. Re-engineering was originated in the world of hardware systems. System is re-engineered to bring down the maintenance cost of the system and increase its life span. Re-engineering is more suitable for software systems as software systems do not wear or tear out. Base is not weakened and there is no
depreciation in case of software systems as it happens in all other hardware systems. Re-engineering of software systems has more scope in the coming future.

Re-engineering cost increases with the age of the system. Every system ages with time and die (a system which is unable to operate or unable to cope up with our changed business process is dead system). Re-engineering enhances the performance and reduces the maintenance cost of the system. Re-engineered system will be more effective, efficient and responsive. Once the system is reengineered it is new system for the organization. Re-engineering does not mean repair (mending) of the system. The system will have new base design, new technology, enhanced requirements, and new platform. Modification of the system is done on the basis of new technology, new brains (human beings), and new environment. Re-engineering is of great importance because it reduces system’s evolution risk, reduce investment, makes system easier to understand and change.

In early years, re-engineering of the software was not acknowledged, the focus was on to discover the new ways to create better software. There was no attention to the old software systems that were getting outdated with the fast changes in the business processes and technology. Every time new developed software was needed to accommodate new business needs. That was the period of software shortage as with small changes in requirements, consideration remained on new development.

In the early 1990 the scenario changed, focus was shifted from developing new software to re-engineering the old legacy software systems. Much attention was given to re-engineering business processes and software. Due to inexperience and unskilled people and lack of tools in field of re-engineering, many projects failed at that time and it was the turning point for re-engineering boom.

Since more than 50% of the total cost of the software development goes to the maintenance of the software. Therefore re-engineering, reorganization and redesign of the software is necessary to reduce the maintenance cost. This factor took the side of re-engineering and made the re-engineering community active in this field.
1.3 Re-engineering Process

Re-engineering process is all the activities which complete the re-engineering of software system. Re-engineering process has three major stages namely (1) Reverse engineering (2) Architecture transformation (3) Forward engineering.

(1) Reverse engineering

In the first stage (Reverse engineering) software is analyzed and modules are separated to know its underlying technology. Reverse engineering involves in extracting design artifacts and making abstractions at higher levels that are easy to understand. It is a process of inspection to know the underlying technology of the software system. In this stage, software engineers move from code level to higher level of abstractions. Some objects are added and/or deleted according to changed requirements and changed business process. It is on left side upward step as shown in the figure 1.4

(2) Architecture transformation

In the second stage, new architecture of the software is designed according to the changes required. Output of the first stage (reverse engineering) will be used as
input for the second stage. Recovered design from reverse engineering is modified according to a new set of requirements. It is architecture transformation stage. This depends upon the changed business process and/or other factors relating to the change in environment or technology. It is top horizontally right ward step as shown in the figure 1.4

(3) **Forward engineering**

In this stage, we move from higher level of abstractions to code level and software is integrated according to new design. It is vertically downward step as shown in the fig. 1.4. Forward engineering means developing the system from design specifications recovered after reverse engineering. In forward engineering software engineers come down from higher level of abstractions to code level development. System is reconstituted from unwrapped different objects/components. The new target system is created by moving downward through the levels of abstractions, a gradual decrease in the abstraction levels of system representation by successive replacement of existing system information with more detailed information.

Re-engineering = Reverse engineering + Enhancement and/or design improvement + Forward engineering

1.4 **Characteristics of Re-engineering**

Re-engineering of software systems is done to meet the contemporary demands of quality, better services, flexibility and low cost. Reengineered software is radically customer oriented and easy to use. Re-engineering and evolution are difficult and expensive. In maintenance system is repaired and in re-engineering system is made new (as new development) and another life span of software starts. In re-engineering business process is changed, requirements are updated, architecture is modified, coding is done according to transformed architecture and run on new platform. Business process is continually changed as activities of process change with changes in technology. Business process is set of logically related activities to achieve a required business outcome. Information Technology is employed to redesign the business processes. It is depicted as in the following the figure. There is close and recursive relationship between Information Technology and BPR (Business Process Re-engineering).
In re-engineering we go from code level to higher level of abstractions where all the changes are made and again we come down from higher level abstractions to the code level (implementation). In re-engineering process, software goes through three forms of a system as shown below in the figure 1.6.

Re-engineered software system works with reduced execution time, better service in quality and quantity and with less maintenance cost.

Advanced software technology plays very important role in re-engineering object oriented software systems. The role of software technology is to make software tools and techniques to speed up and easy re-engineering process. Software tools are available for understanding software, improving business process, improving software architecture and developing software. In the coming future more sophisticated tools will be available for re-engineering. 100 % automation in re-
engineering is not possible but step wise and process wise automation can be achieved up to great extent.

In this modern age of information technology, software is needed from small to big businesses. More and more software is needed. More and more software engineers and developers are needed.

The technology is changing fast and with this technology change, business processes are changing, work platforms are changing, software languages are coming more sophisticated. New software tools and techniques are coming in the market.

With speedy changes in every aspect around software, software needs change. These changes are done by software maintainers. Accumulative effects of minor changes make the software complex and after some time it will be difficult to maintain the software. Then software is re-engineered. As software exist everywhere, re-engineering of software is needed at mass level. Hence the scope of re-engineering is good in coming future.

When maintenance is difficult and/or costly, legacy software is reengineered or retired (thrown away). If the software is retired, costly new software will be purchased by the client company. This routine can happen when small changes are required in the software. It will be big financial loss to the organizations. The problem of software backlog will also arise.

1.5 Purposes and Objectives of Re-engineering

Re-engineering is the only way to utilize the software fully and solve the problem of software backlogs. Life span of software can be increased by re-engineering of software system. Systems inevitably grow old (becomes complex) and die (retire) but software system can be rejuvenated through re-engineering. In this age of fast changes, technology and process models are changing fast. Re-engineering of software improves the performance, efficiency and reliability of the software. Re-engineering makes the system new to fit in the new set of requirements, novel technology and new environment. Re-engineering brings down the maintenance cost of the software to the normal level for another life span of
software. Maintenance of the software starts after the delivery of the software to correct faults, enhance performance and other attributes of the software system. Continually software maintenance impacts the quality of the software and maintenance cost increases. After few years when maintenance cost is too high, we think of re-engineering. Following are the reasons for high maintenance cost.

1. Original developers no longer available
2. Outdated development methods used
3. Ripple effects of patches and modifications
4. Missing or outdated documentation.
5. Further evolution and development is prohibitively expensive

When maintenance cost is not tolerable software users have to go for re-engineering or retiring the software because maintenance cost increases rapidly when software is aged. This work will be important for both users of software and reengineers. Jobs of the engineers will be comfortable with re-engineering model(s) presented in the work. More software engineers will be attracted to this field, more research will be done. More tools and techniques will be developed. There will be software re-engineering boom. Software will be fully utilized and there will not be wastage of resources (costly software). Problem of software backlog will also be solved.

In general, following are the purposes and objectives of re-engineering object oriented software systems.

- Reduce maintenance cost
- Functional enhancement
- Improve maintainability
- Improve customer satisfaction
- Better language or hardware platform
- Improve reliability
- Increasing software age
1.6 Objectives of the Present Work

In this research work, efforts are made to explore and design the re-engineering model(s) for object-oriented software systems. In this thesis, a methodology and a set of techniques are proposed to assist software engineers and managers migrating object-oriented legacy systems. The area of re-engineering of software systems does not exist from centuries, it is new area of research and not much developed. This area has enormous importance as software is increasing every where and with fast changes re-engineering is also needed in parallel. At the same time re-engineering is expensive and complex. The people go for maintenance or new product (software system) instead of re-engineering.

Re-engineering can be made popular, easy, and less expensive by innovations in re-engineering area. Highly sophisticated software tools, techniques, and precise model(s) can be developed for this purpose.

New model(s) are added to re-engineering field. Model(s) are designed for all the processes of complete re-engineering cycle. Re-engineering cycle is period from identification of the software system for re-engineering to implementation of the re-engineered software.

1.7 Motivation for this work

Motivation behind this research area is mainly software cost and wastage of software resources. In the software industries, old existing software systems must be made to last longer to reduce the software investment. Re-engineering is two way benefits on one hand re-engineering cost and time is significantly less than new development and on the other performance and other aspects of the system will be increased. Re-engineering reduces the maintenance cost and hence less people are required for maintenance and consequently software productivity will be increased. Software structure can be improved with latest technology through re-engineering, and then software changes will not only be easier but also less error-prone. This will result in less rework and hence productivity will be increased.

Legacy software systems exist everywhere, some time maintenance cost becomes unbearable then re-engineering is the only option to reduce the
maintenance cost and to avoid costly new development. Re-engineering can reduce the overall investments on software in the organizations.

Why specially focused on re-engineering of object oriented software systems? There are considerable object oriented software systems, although they are not old to become legacy systems. But it is the rate at which they have been developed and adopted without having been reengineered. Object-oriented development paradigm has been adopted by many software organizations. In future, software development trends will be towards components, frames and patterns etc. There will be heavy rush in re-engineering of object oriented software systems and at the same time such systems are more suitable for re-engineering. Because objects and patterns based software stems are more reusable and understandable.

This research attempted to offer model(s) to select the alternative of re-engineering and to act upon re-engineering.

The software systems which never depreciate and/or never tear out can be utilized more effectively and can work for more time with re-engineering.

With the help of model(s) presented in this work, re-engineering of object-oriented software systems will be aggravated.

1.8 Organization of the Thesis

The entire study is divided into seven chapters. All the chapters are roped in a systematic way to understand and utilize the work swiftly. A brief description of each of these chapters is as given below:

Chapter-I

The first chapter is introductory chapter in which fundamentals of the re-engineering are discussed. All the words in title of the thesis are explained and the cause of taking this topic is discussed. Basics of objects, object oriented software systems are talked about. Software engineering, software re-engineering, process of re-engineering were discussed. The need of re-engineering, characteristics of re-engineering, purposes and objectives of re-engineering and motivation for this work are discussed in this chapter.
Chapter-II

Literature available on this topic is reviewed and discussed in this chapter. This helped me to reach at the cutting edge of the knowledge on this topic. Different sources like books, e-books, national and international journal, conference papers, review articles, discussions with software engineers, and software users are utilized. Internet and other networks are also used as a source of literature review.

Chapter-III

In this chapter research methodology used is explained. To select research methodology depends upon the nature of problem. In present problem mixed research methodology (qualitative and quantitative) was used. Literature was reviewed on the topic and case studies were conducted on status and problems in the way of re-engineering. Various parameters of software systems were compared before and after re-engineering. Exploratory research was also used which help to diagnose a situation and thinking of different alternatives to discover new ideas and concepts.

Chapter-IV

In this chapter existing re-engineering model(s), tools and techniques were studied thoroughly to think for better models. Many model(s) were old fashioned and were applicable to software systems in procedural oriented languages. Models do not exist for some processes in re-engineering creating gap in complete re-engineering process. Common existing models up to date were discussed in this chapter. Their working, applications and limitations in the real situations were also discussed.

Chapter-V

Model(s) are proposed for various phases of re-engineering process. These model(s) are applicable particularly for re-engineering object-oriented software systems. Models are useful for software engineers (software development companies) and software managers (client companies). Proposed model(s) will help the software managers to understand the confusing terms and concepts in subject area of re-engineering. Candidate software can be identified for re-engineering and
cost of re-engineering can be estimated. Models are proposed to know the right time for re-engineering and the techniques for re-engineering.

In this chapter, it is tried to eliminate the gaps left by existing models.

Chapter-VI

In this chapter case study results are presented. Case studies were conducted on status and problems in re-engineering. Software improvement and enhancement in various aspects of software by re-engineering were estimated. Data were collected, analyzed and reports and graphs were generated. Results are based on empirical data and represent the real situations in the software industries.

Chapter-VII

Conclusion and recommendations based on the above study are mentioned in this chapter. The chapter contains gist of the whole study and real picture of re-engineering of software systems. Findings of the study are the main highlights of the chapter. It also contains the derived facts and figures. The chapter includes suggestions and directives for future work that can be carried out further from this threshold.