Reengineering

4.1 General Idea

- Reengineering is the examination, analysis, and restructuring of an existing software system to reconstitute it in a new form, and the subsequent implementation of the new form.
- Example: initially Unix was developed in assembly language. When language C came into existence, Unix was re-engineered in C, because working in assembly language was difficult.
- The goal of reengineering is to:
  - understand the existing software system artifacts, namely, specification, design, implementation, and documentation, and
  - improve the functionality and quality attributes of the system.

4.1 General Idea

- Software systems are re engineered by keeping one or more of the following four general objectives in mind:
  - Improving maintainability.
  - Migrating to a new technology.
  - Improving quality.
  - Preparing for functional enhancement.

4.2 Reengineering Concepts

- **Abstraction** and **Refinement** are key concepts used in software development, and both the concepts are equally useful in reengineering.
- It may be recalled that abstraction enables software maintenance personnel to reduce the complexity of understanding a system by:

  (i) focusing on the more significant information about the system; and

  (ii) Hiding the irrelevant details at the moment.
- On the other hand, refinement is the reverse of abstraction.

Principle of abstraction: The level of abstraction of the representation of a system can be gradually increased by successively replacing the details with abstract information. By means of abstraction one can produce a view that focuses on selected system characteristics by hiding information about other characteristics.

Principle of refinement: The level of abstraction of the representation of the system is gradually decreased by successively replacing some aspects of the system with more details.

4.2 Reengineering Concepts

- The concepts of abstraction and refinement are used to create models of software development as sequences of phases, where the phases map to specific levels of **abstraction** or **refinement**. as shown in Figure 4.1.
- The four levels are:
  - Conceptual,
  - Requirements,
  - Design, and
  - Implementation.

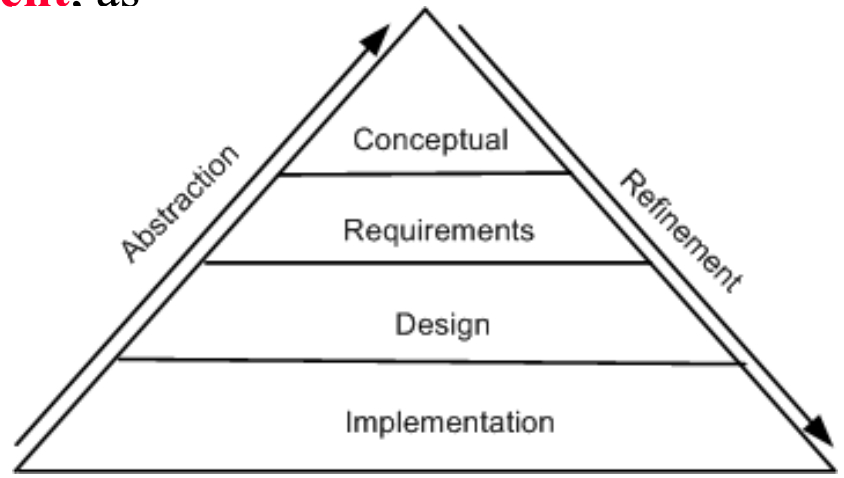

Figure 4.1 levels of abstraction and refinement
© IEEE, 1992

- The refinement process:

  why? ! what? ! what & how? ! how?

- The abstraction process:

  how? ! what & how? ! what? ! why?

4.2 Reengineering Concepts

- An optional principle called **alteration** underlies many reengineering methods.
- **Principle of alteration:** The alteration principle refers to the conduction of one or more changes in system abstraction without changing the level.
- **Reengineering principles** are represented by means of arrows. Abstraction is represented by an up-arrow, alteration is represented by a horizontal arrow, and refinement by a down-arrow.
- The arrows depicting refinement and abstraction are slanted, thereby indicating the increase and decrease, respectively, of system information.
- It may be noted that alteration is non-essential for reengineering.

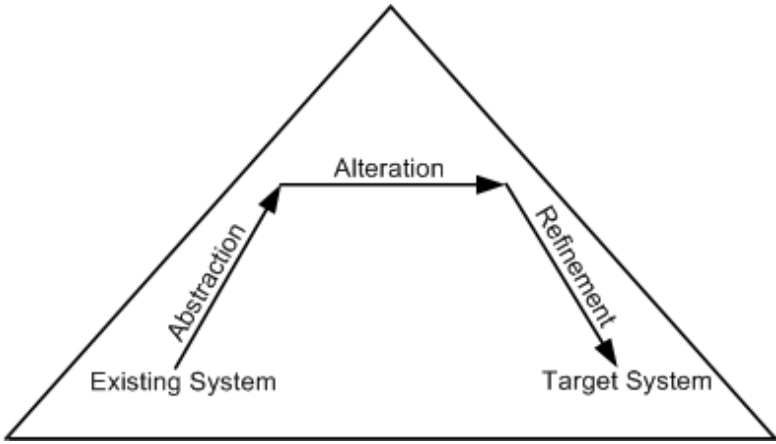

Figure 4.2 Conceptual basis for the reengineering process © IEEE, 1992

4.3 A General Model For Software Reengineering

- The reengineering process accepts as input the existing code of a system and produces the code of the renovated system.
- The reengineering process may be as straightforward as translating with a tool the source code from the given language to source code in another language.
- For example, a program written in BASIC can be translated into a new program in C.
- The reengineering process may be very complex as explained below:
  - recreate a design from the existing source code.
  - find the requirements of the system being reengineered.
  - compare the existing requirements with the new ones.
  - remove those requirements that are not needed in the renovated system.
  - make a new design of the desired system.
  - code the new system.

4.3 A General Model For Software Reengineering

- The model in the figure proposed by Eric J. Byrne suggests that reengineering is a sequence of three activities:
  - reverse engineering, re-design, and forward engineering
  - strongly founded in three principles, namely, abstraction, alteration, and refinement.

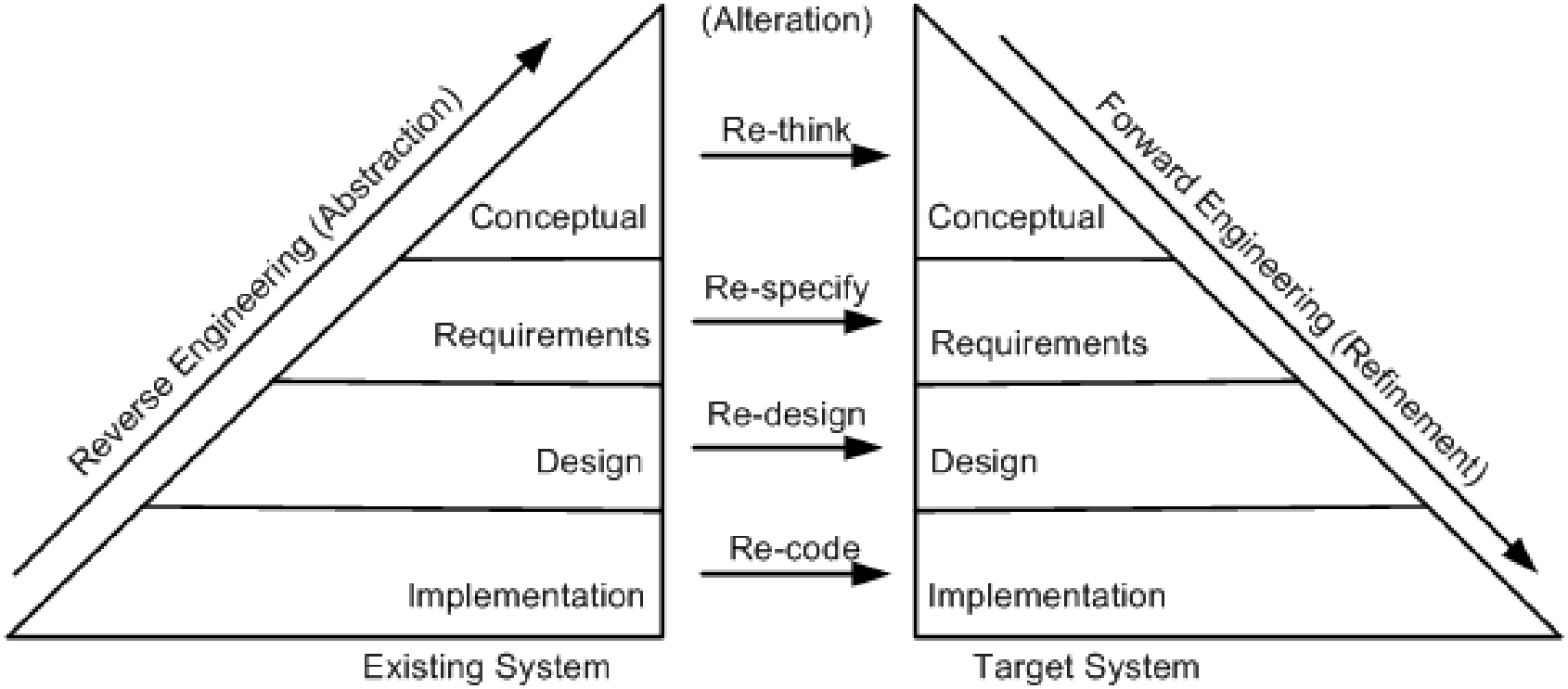

Figure: General model of software reengineering © IEEE, 1992

4.3 A General Model For Software Reengineering

- A visual metaphor called horseshoe, as depicted in Figure 4.4, was developed by Kazman et al. to describe a three-step architectural reengineering process.
- Three distinct segments of the horseshoe are the left side, the top part, and the right side. Those three parts denote the three steps of the reengineering process.

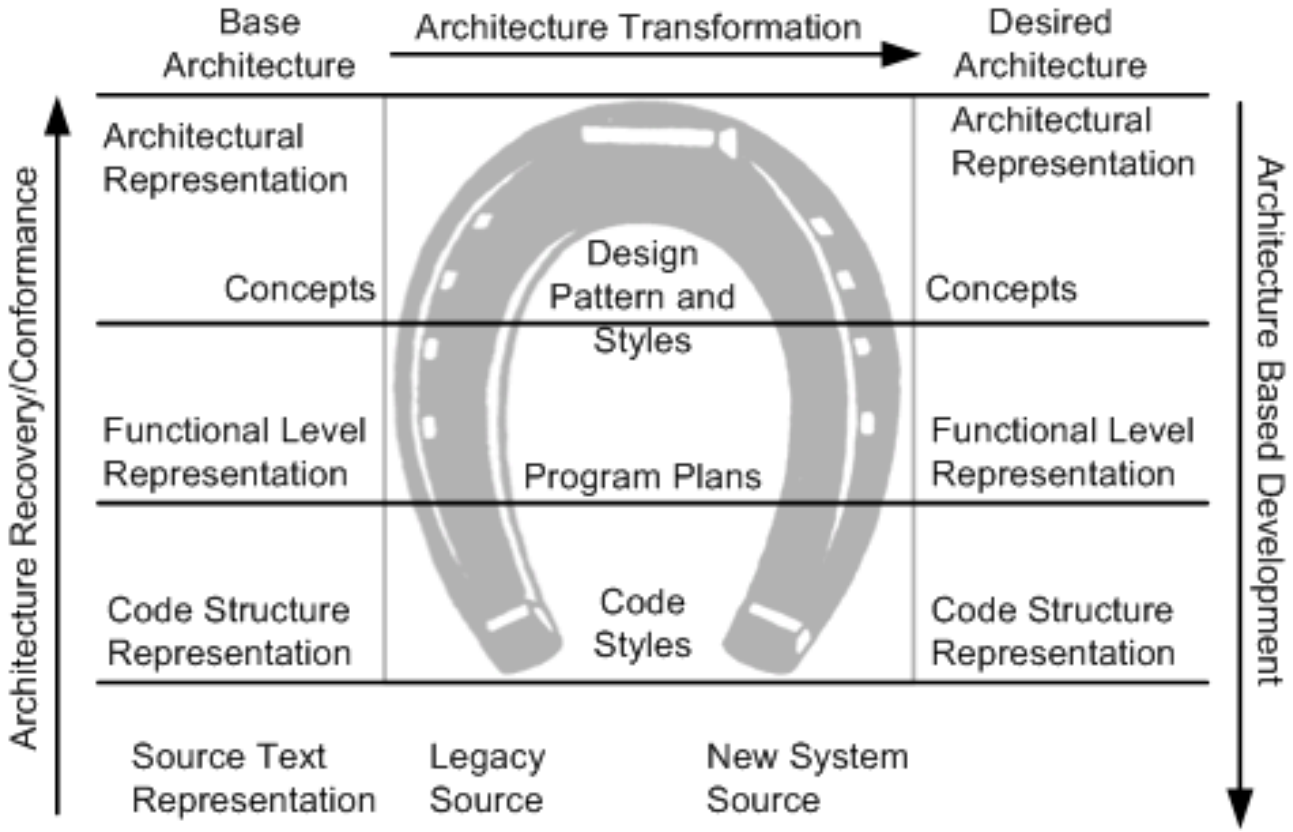

Figure 4.4 Horseshoe model of reengineering © IEEE, 1998

4.3 A General Model For Software Reengineering

- In summary, it is evident that reengineering entails:

  (i) the creation of a more abstract view of the system by means of some reverse engineering activities,

  (ii) the restructuring of the abstract view, and

  (iii) implementation of the system in a new form by means of forward engineering activities.

- This process is formally captured by Jacobson and Lindstorm with the following expression:

  Reengineering = Reverse engineering + Δ + Forward engineering.

- The element "Δ" captures alterations made to the original system.
- Two major dimensions of alteration are: change in functionality and change in implementation technique.
- A change in functionality comes from a change in the business rules,
- Next, concerning a change of implementation technique, an end-user of a system never knows if the system is implemented in an object-oriented language or a procedural language.

4.3.1 Types of Change

Based on the type of changes required, system characteristics are divided into groups: **rethink, respecify, redesign**, and **re-code**.

Recode:

- Implementation characteristics of the source program are changed by re-coding it. Source-code level changes are performed by means of rephrasing and program translation.
- In the latter approach, a program is transformed into a program in a different language. On the other hand, rephrasing keeps the program in the same language
- Examples of translation scenarios are **compilation, decompilation**, and **migration**.
- Examples of rephrasing scenarios are **normalization**, **optimization, refactoring**, and **renovation.**

Redesign:

- The design characteristics of the software are altered by re-designing the system. Common changes to the software design include:

  (i) restructuring the architecture;

  (ii) Modifying the data model of the system; and

  (iii) replacing a procedure or an algorithm with a more efficient one.

4.3.1 Types of Change

Respecify:

- This means changing the requirement characteristics of the system in two ways:

  (i) change the form of the requirements, and

  (ii) change the scope of the requirements.

Rethink:

- Re-thinking a system means manipulating the concepts embodied in an existing system to create a system that operates in a different problem domain.
- It involves changing the conceptual characteristics of the system, and it can lead to the system being changed in a fundamental way.
- Moving from the development of an ordinary cellular phone to the development of smartphone system is an example of Re-think.

4.3.2 Software Reengineering Strategies

Three strategies that specify the basic steps of reengineering are **rewrite**, **rework**, and **replace**.

Rewrite strategy:

This strategy reflects the principle of alteration. By means of alteration, an operational system is transformed into a new system, while preserving the abstraction level of the original system. For example, the Fortran code of a system can be rewritten in the C language.

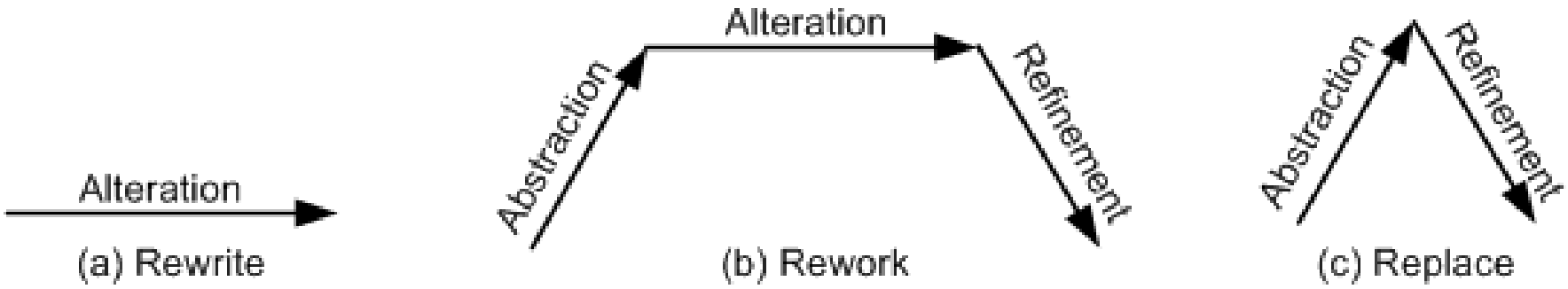

Figure 4.5 Conceptual basis for reengineering strategies © IEEE, 1992

4.3.2 Software Reengineering Strategies

Rework strategy:

- The rework strategy applies all the three principles.
- Let the goal of a reengineering project is to replace the unstructured control flow constructs, namely GOTOs, with more commonly used structured constructs, say, a "for" loop.
- A classical, rework strategy based approach is as follows:
  - Application of abstraction: By parsing the code, generate a control-flow graph (CFG) for the given system.
  - Application of alteration: Apply a restructuring algorithm to the control-flow graph to produce a structured control-flow graph.
  - Application of refinement: Translate the new, structured control-flow graph back into the original programming language.

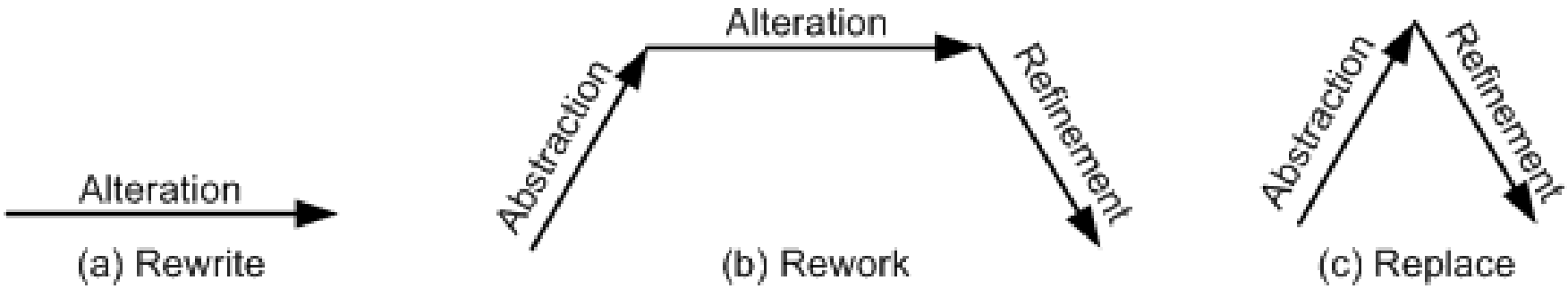

Figure 4.5 Conceptual basis for reengineering strategies © IEEE, 1992

4.3.2 Software Reengineering Strategies

Replace strategy:

- The replace strategy applies two principles, namely, abstraction and refinement.
- To change a certain characteristic of a system:
  - (i) the system is reconstructed at a higher level of abstraction by hiding the details of the characteristic; and
  - (ii) a suitable representation for the target system is generated at a lower level of abstraction by applying refinement.
- Let us reconsider the GOTO example. By means of abstraction, a program is represented at a higher level without using control flow concepts.
- Next, by means of refinement, the system is represented at a lower level of abstraction with a new structured control flow.

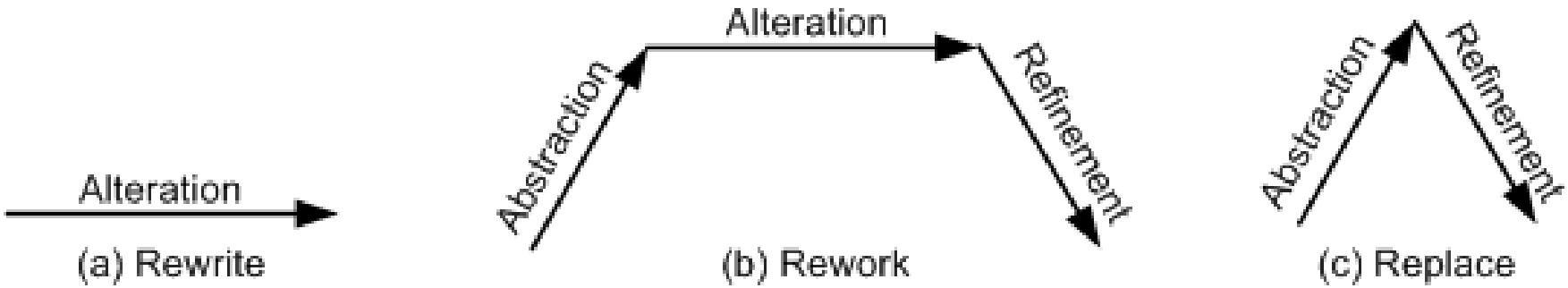

Figure 4.5 Conceptual basis for reengineering strategies © IEEE, 1992

4.4 Reengineering Process

- An ordered set of activities designed to perform a specific task is called a process.
- For ease of understanding and communication, processes are described by means of process models.
- For example, in the software development domain, the Waterfall process model is widely used in developing well-understood software systems.
- Process models are used to comprehend, evaluate, reason about, and improve processes.
- Intuitively, process models are described by means of important relationships among data objects, human roles, activities, and tools.
- We will discuss five process models for software reengineering.
- The five approaches are different in two aspects:

(i) the extent of reengineering performed, and

(ii) the rate of substitution of the operational system with the new one.

4.4.1 Reengineering Approaches

Big Bang Approach

- The "**Big Bang**" approach replaces the whole system at once.
- Once a reengineering effort is initiated, it is continued until all the objectives of the project are achieved and the target system is constructed.
- This approach is generally used if reengineering cannot be done in parts.
- For example, if there is a need to move to a different system architecture, then all components affected by such a move must be changed at once.
- The consequent advantage is that the system is brought into its new environment all at once.
- The disadvantage of Big Bang is that the reengineering project becomes a monolithic task, which may not be desirable in all situations.
- In addition, the Big Bang approach consumes too much resources at once for large systems, and takes a long stretch of time before the new system I visible.

4.4.1 Reengineering Approaches

Incremental Approach

- In this approach a system is reengineered gradually, one step closer to the target system at a time.
- For a large system, several new interim versions are produced and released.
- Successive interim versions satisfy increasingly more project goals than their preceding versions.
- The advantages of this approach are as follows:

  (i) locating errors becomes easier, because one can clearly identify the newly added components, and

  (ii) It becomes easy for the customer to notice progress, because interim versions are released.
- The disadvantages of the incremental approach are as follows:

  (i) with multiple interim versions and their careful version controls, the incremental approach takes much longer to complete, and

  (ii) even if there is a need, the entire architecture of the system cannot be changed.

4.4.1 Reengineering Approaches

Partial Approach

- In this approach, only a part of the system is reengineered and then it is integrated with the non-engineered portion of the system.
- One must decide whether to use a "Big Bang" approach or an "Incremental" approach for the portion to be reengineered.
- The following three steps are followed in the partial approach:
  - In the first step, the existing system is partitioned into two parts: one part is identified to be reengineered and the remaining part to be not reengineered.
  - In the second step, reengineering work is performed using either the "Big Bang" or the "Incremental" approach.
  - In the third step, the two parts, namely, the not-to-be-reengineered part and the reengineered part of the system, are integrated to make up the new system.
- The partial approach has the advantage of reducing the scope of reengineering that is less time and costs less.
- A disadvantage of the partial approach is that modifications are not performed to the interface between the portion modified and the portion not modified.

4.4.1 Reengineering Approaches

Iterative Approach

- The reengineering process is applied on the source code of a few procedures at a time, with each reengineering operation lasting for a short time.
- This process is repeatedly executed on different components in different stages.
- During the execution of the process, ensure that the four types of components can coexist:
  - old components not reengineered,
  - components currently being reengineered,
  - components already reengineered, and
  - new components added to the system.
- There are two advantages of the iterative reengineering process:

  (i) it guarantees the continued operation of the system during the execution of the reengineering process, and

  (ii) the maintainers' and the users' familiarities with the system are preserved.
- The disadvantage of this approach is the need to keep track of the four types of components during the reengineering process.
- In addition, both the old and the newly reengineered components need to be maintained.

4.4.1 Reengineering Approaches

Evolutionary Approach

- In the ”Evolutionary” approach components of the original system are substituted with re-engineered components.
- In this approach, the existing components are grouped by functions and reengineered into new components.
- Software engineers focus their reengineering efforts on identifying functional objects irrespective of the locations of those components within the current system.
- As a result, the new system is built with functionally cohesive components as needed.
- There are two advantages of the “Evolutionary” approach:

  (i) the resulting design is more cohesive, and

  (ii) the scope of individual components is reduced.
- A major disadvantage:

  (i) all the functions with much similarities must be first identified throughout the operational system.

  (ii) next, those functions are refined as one unit in the new system.