Chapter 6

Software Re-Engineering
&
Reverse Engineering

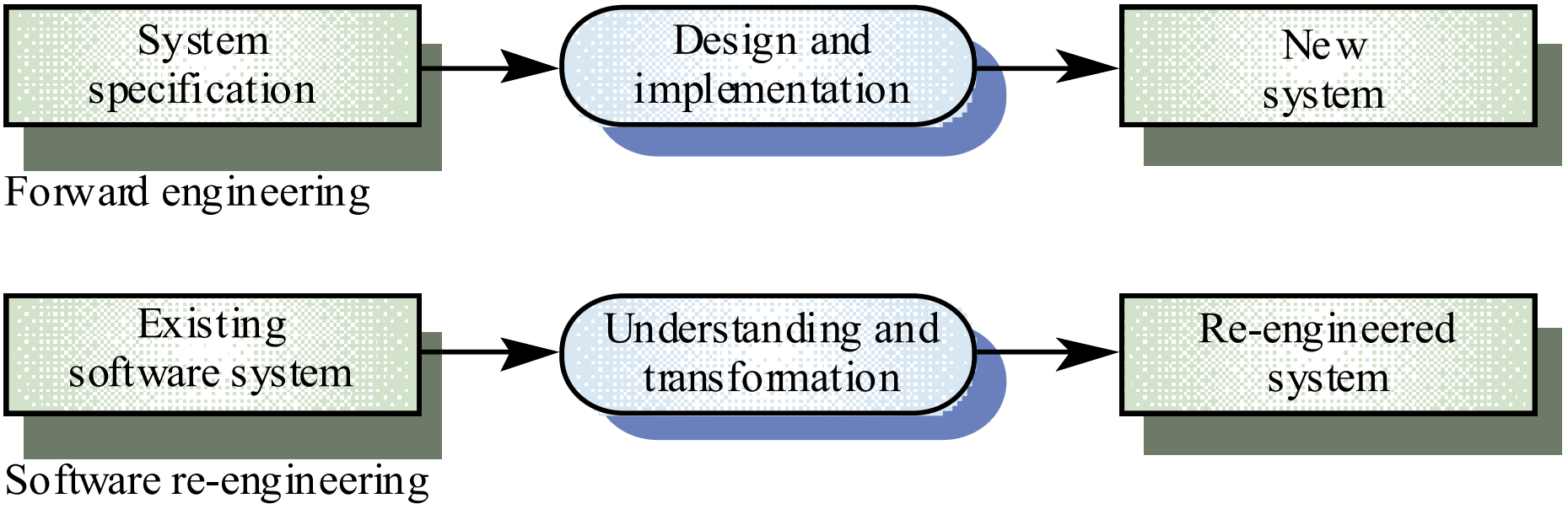
Definitions

- **Forward engineering** - traditional software engineering approach starting with requirements analysis and progressing to implementation of a system
- **Reverse engineering** – system analysis process to:
  - identify the system's components and their interrelationships and
  - create representations of the system in another form or at higher levels of abstraction
- **Reengineering** - process of analysis and change whereby a system is modified by first reverse engineering and then forward engineering.
- **Re-factoring** (restructuring) - transformation of a system from one representational form to another

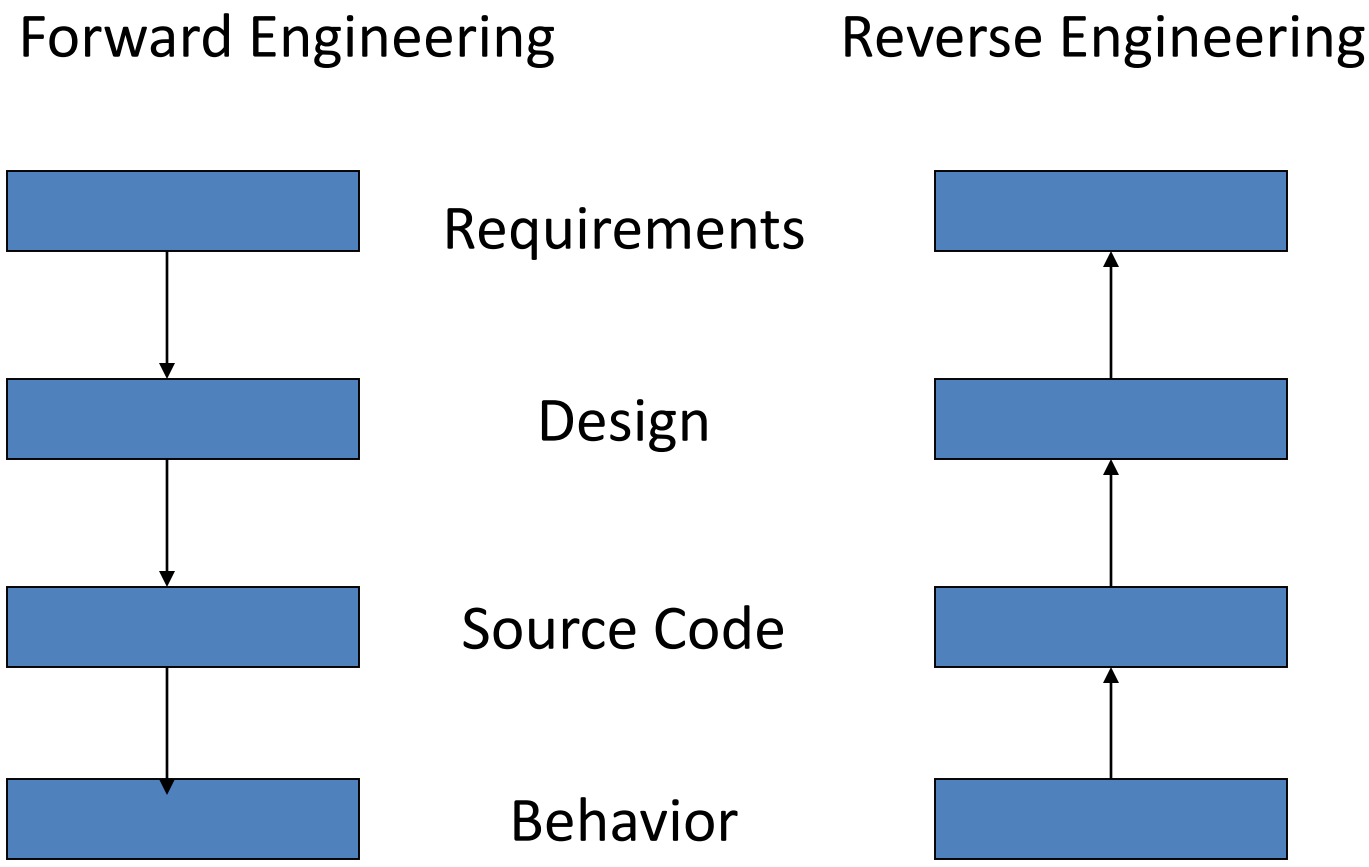
Reengineering = **Reverse engineering+ Δ + Forward engineering**

Δ=alterations

Forward Engineering and Reengineering



What is Reverse Engineering ?



Goals of Reengineering

- Port to other Platform
  - when platform support becomes obsolete
- Design extraction
  - to improve maintainability, portability, etc.
- Exploitation of New Technology
  - new language features, standards, libraries, etc.
  - when tools to support restructuring are readily available

Re-engineering advantages

- Reduced risk
  - There is a high risk in new software development: development problems, staffing problems and specification problems
- Reduced cost
  - Cost of re-engineering is often less than costs of developing new software

Reengineering Techniques

- **Restructuring**
  - is the transformation from one representation form to another at the same relative abstraction level, while preserving the system's external behavior
  - source code translation
- **Data Reengineering**
  - integrating and centralizing multiple databases
  - unifying multiple, inconsistent representations
  - upgrading data models
- **Refactoring**
  - is restructuring within an object-oriented context
  - Misuse of inheritance
    - change inheritance to delegation if the subclass doesn't use attributes
  - Missing inheritance
    - duplicated code, and case statements to select behavior
  - Misplaced operations
    - unexploited cohesion — operations outside instead of inside classes

Types of Restructuring

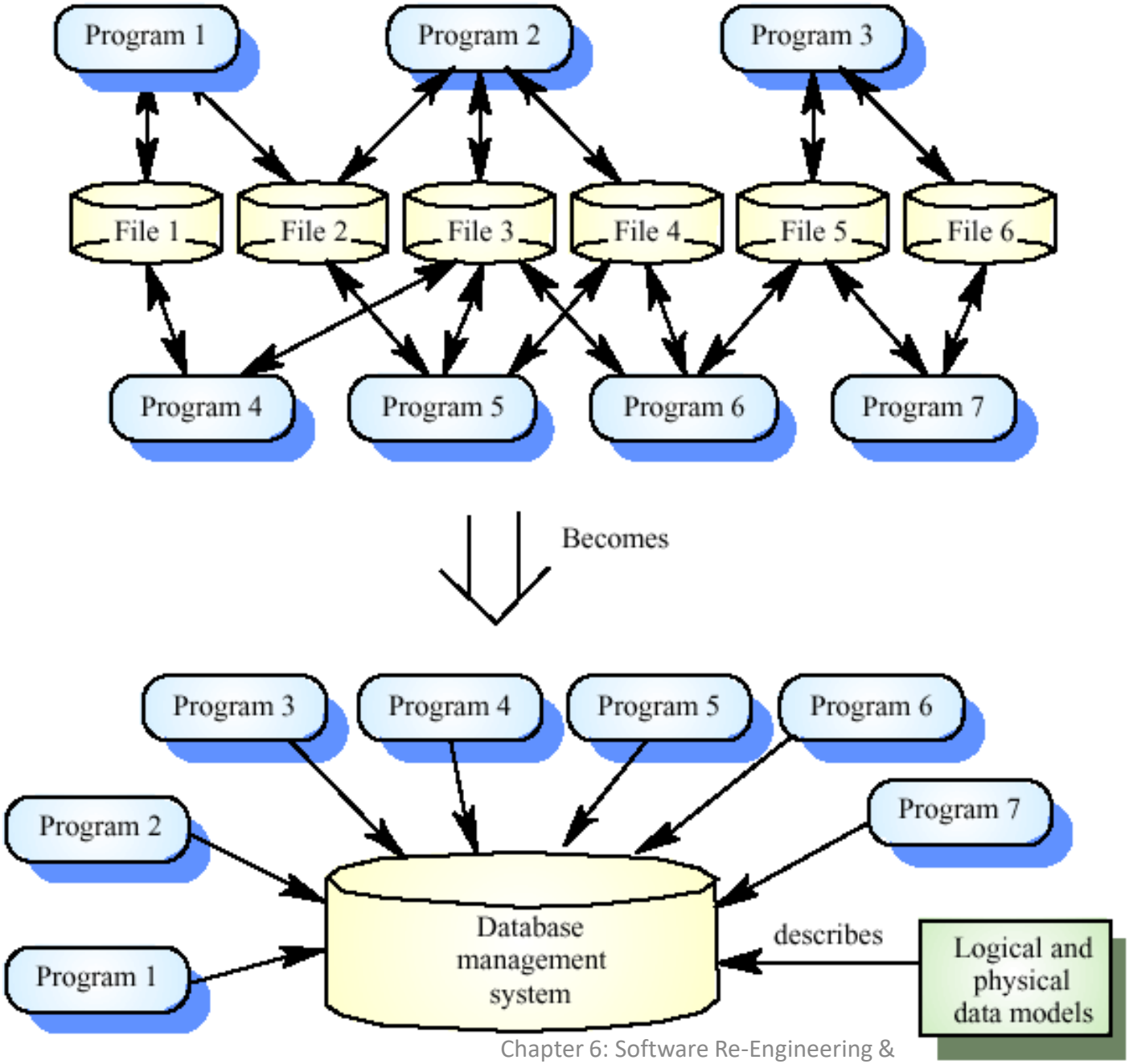
- Code restructuring
  - Program transformation
  - Architecture transformations
- Data restructuring
  - analysis of source code
  - data redesign
  - file or database translation

Approaches to data restructuring

Approach	Description
Data cleanup	The data records and values are analysed to improve their quality. Duplicates are removed, redundant information is deleted and a consistent format applied to all records. This should not normally require any associated program changes.
Data extension	In this case, the data and associated programs are re-engineered to remove limits on the data processing. This may require changes to programs to increase field lengths, modify upper limits on the tables, etc. The data itself may then have to be rewritten and cleaned up to reflect the program changes.
Data migration	In this case, data is moved into the control of a modern database management system. The data may be stored in separate files or may be managed by an older type of DBMS.

Data problems

- End-users want data on their desktop machines rather than in a file system. They need to be able to download this data from a DBMS
- Redundant data may be stored in different formats in different places in the system
- Systems may have to process much more data than was originally intended by their designers



Data migration

Contd..

- **Data naming problems**
  - Names may be hard to understand.
  - The same data may have different names in different programs
- **Field length problems**
  - The same item may be assigned different lengths in different programs
- **Record organisation problems**
  - Records representing the same entity may be organised differently in different programs
- **Hard-coded literals**
- **No data dictionary**

Approaches for Reengineering

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 **advantage** is that the system is brought into its new environment all at once.
- The **disadvantage** is that it consumes too much resources at once for large systems and takes a long stretch of time before the new system is visible.

Incremental approach

- *A* **system** is reengineered **gradually, one step closer to the target system at a time**.
- Thus, 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 desired system is said to be generated after all the project goals are achieved.
- 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.

Cont'd

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.

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.

Contd..

- The **advantage** of reducing the scope of reengineering to a level that best matches an organization's current need and desire to spend a certain amount of resources.
- A reduced scope implies that the selected portions of a system to be modified are those that are urgently in need of reengineering.
- A reduced scope of reengineering takes 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.

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.
- Their coexistence is necessary for the operational continuity of the system.

Contd..

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.

Evolutionary approach

- *Similar to the "Incremental" approach, in the "Evolutionary"* approach **components of the original system are substituted with reengineered components.**
- However, 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.**

Contd..

- **The advantages of the "Evolutionary" approach:**

  (i) the resulting design is more cohesive and

  (ii) the scope of individual components is reduced.

- **A major disadvantage of the approach is as follows:**
  - all the functions with much similarities must be first identified throughout the operational system; next,
  - Those functions are refined as one unit in the new system.

Principles of Reverse Engineering

- **Reverse Engineering:**
  - Systematic process of acquiring important design factors and information regarding engineering aspects from an existing product
  - A process which analyses a product/technology to find out the design aspects and its functions
  - A kind of analysis which engages an individual in a process of *constructive learning* of design and its functionality of systems and products

Reverse Engineering

- **Goal**: to facilitate change by allowing a software system to be understood in terms of **what it does**, **how it works** and its **architectural representation**.
- **Objectives**:
  - to recover lost information,
  - to facilitate migration between platforms,
  - to improve and/or provide new documentation,
  - to extract reusable components,
  - to reduce maintenance effort,
  - to cope with complexity,
  - to develop similar or competitive products.

Reverse Engineering Concepts

- **Abstraction level**
  - ideally want to be able to derive design information at the highest level possible
  - As the abstraction level increases, the software engineer is provided with information that will allow easier understanding of the program.
- **Completeness**
  - level of detail provided at a given abstraction level
  - As the completeness decreases as the abstraction level increases
- **Interactivity**
  - degree to which humans are integrated with automated reverse engineering tools
  - as the abstraction level increases, interactivity must increase

Contd..

- **Directionality**
  - one-way means the software engineer doing the maintenance activity is given all information extracted from source code
  - two-way means the information is fed to a reengineering tool that attempts to regenerate the old program
- **Extraction of abstractions**
  - meaningful specification of processing performed is derived from old source code

Reverse Engineering Activities

- Understanding process
  - source code is analyzed to at varying levels of detail
  - to understand procedural abstractions and overall functionality
- Understanding data
  - internal data structures
  - database structure
- Understanding user interfaces
  - what are basic actions processed by the interface?
  - what is system's behavioral response to these actions?
- Reverse Engineering Techniques
  - Re-documentation
  - Design recovery

Factors that Motivate the Application of Reverse Engineering

Indicator		Motivation
1.	Missing or incomplete design/specification	Product / environment related
2.	Out-of-date, incorrect or missing documentation	
3.	Increased program complexity	
4.	Poorly structured source code	
5.	Need to translate programs into a different programming language	
6.	Need to make compatible products	
7.	Need to migrate between different software or hardware platforms	
8.	Static or increasing bug backlog	Maintenance process related
9.	Decreasing personnel productivity	
10.	Need for continuous and excessive corrective change	
11.	Need to extend economic life of system	Commercially related
12.	Need to make similar but non-identical product	

Benefits of Reverse Engineering for Software Maintenance

- **Corrective change:**
  - abstraction of unnecessary detail gives greater insight into the parts of the program to be corrected
  - easier to identify defective program components and the source of residual errors
- **Adaptive/perfective change:**
  - Eases understanding of system's components and their interrelationships, showing where new requirements fit and how they relate to existing components
  - Extracted information which can be used during enhancement of the system or for the development of another product

Reverse Engineering Tools

- The process of reverse engineering is accomplished by making use of some tools that are categorized into debuggers or dis-assemblers-
- **Disassemblers** – A disassembler is used to convert binary code into assembly code and also used to extract strings, imported and exported functions, libraries etc.
- The disassemblers convert the machine language into a user-friendly format.
- **Debuggers** – This tool expands the functionality of a disassembler by supporting the CPU registers, the hex duping of the program, view of stack etc.
- Using debuggers, the programmers can set breakpoints and edit the assembly code at run time.
- Debuggers analyze the binary in a similar way as the disassemblers and allow the reverser to step through the code by running one line at a time to investigate the results.

Contd..

- **Decompiler-** A decompiler represents executable binary files in a readable form.
- More precisely, it transforms binary code into text that software developers can read and modify.
- The software security industry relies on this transformation to analyze and validate programs.
- The analysis is performed on the binary code because the source code (the text form of the software) traditionally is not available, because it is considered a commercial secret.

Thank You