Faculty of Computers and Information
BSc. In Software Engineering Program

Dr. Lamia Abo Zaid

د. لمياء أبوزيد

Based on slides by © Tripathy & Naik

Software Evolution : TOC

1. Introduction to Software Maintenance & Evolution
2. Taxonomy of Software Maintenance and Evolution
3. Evolution and Maintenance Models
4. Program Comprehension
5. Impact Analysis
6. Refactoring
7. Reengineering
8. Legacy Information Systems
9. Reuse and Domain Engineering

Reengineering – What and Why ?

❑What is Reengineering ?

- 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.

❑Why do we do reengineering?

1. Understand the existing software system artifacts, namely, specification, design, implementation, and documentation
2. Improve the functionality and quality attributes of the system.

- Some examples of quality attributes are: performance, reliability, correctness, integrity, efficiency, maintainability, usability, testability, interoperability, reusability, and portability

Objectives of Reengineering

Software systems are reengineered to meet one or more of the objectives:

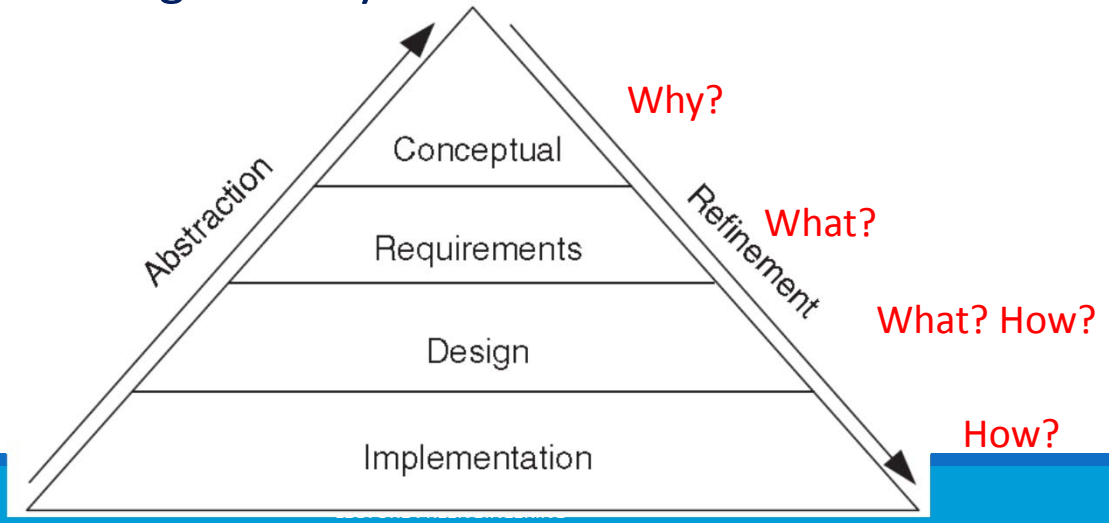
❑ Improve maintainability	Lehman's second law, increasing complexity
❑ Migrate to a new technology	Lehman's first law, continuing change
❑ Improve quality	Lehman's seventh law, declining quality
❑ Prepare for functional enhancement	Lehman's sixth law, continuing growth

Reengineering Concepts

- Abstraction and Refinement are key concepts used in software development, and are equally useful in reengineering.
- Principle of abstraction: By means of abstraction one can produce a view that focuses on selected system characteristics by hiding information about other characteristics. The level of abstraction of the representation of a system can be gradually increased by successively replacing the details with abstract information.
- 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.
- Refinement is the reverse of abstraction

Reengineering Concepts

- A new software is created by going downward from the top, i.e. highest level of abstraction to the bottom i.e. lowest level.
- This downward movement is known as Forward Engineering
- Upward movement through the layers of abstractions is called Reverse Engineering.



Reengineering Concepts

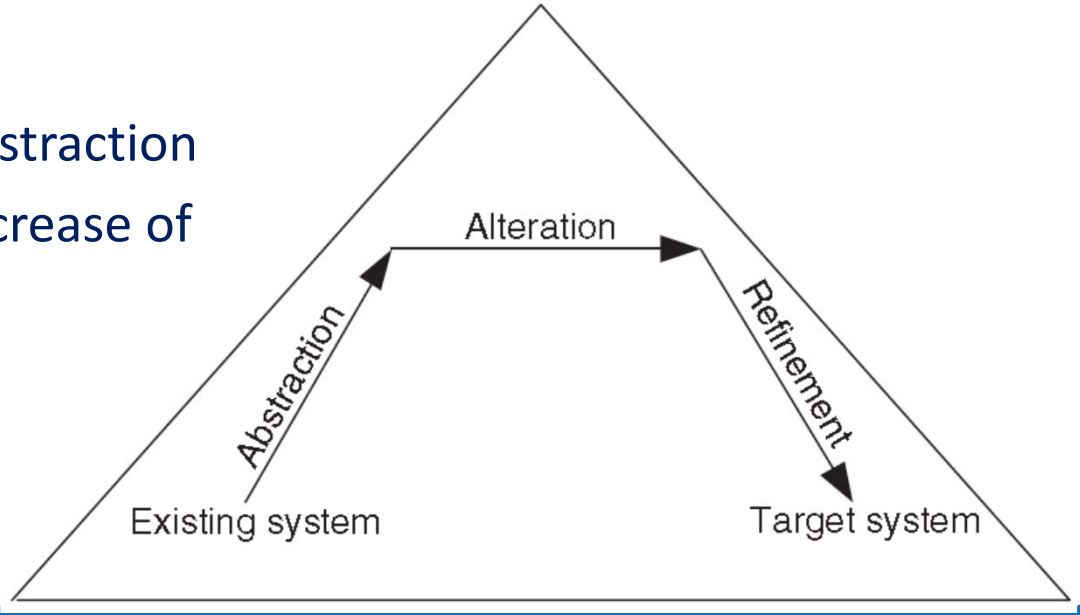
- An optional principle called **alteration** underlies many reengineering methods.
- **Principle of alteration:** The making of some changes to a system representation is known as alteration. Alteration does not involve any change to the degree of abstraction, and it does not involve modification, deletion, and addition of information.

Reengineering Concepts

- **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 sloped indicating the increase and decrease of system information respectively.

- A term closely related to “alteration” is restructuring (refactoring)



General Reengineering Model

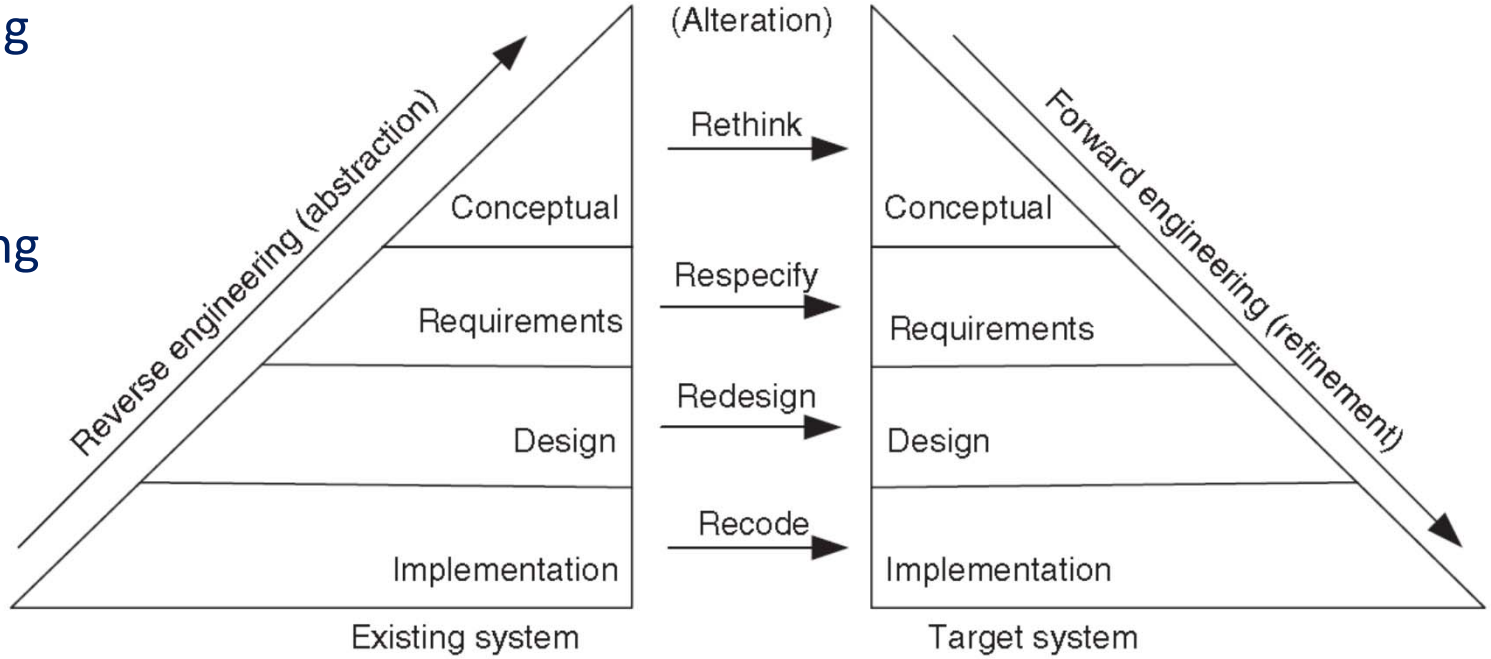
The reengineering process consists of the following stages:

1. Recreate a design from the existing source code.
2. Find the requirements of the system being reengineered.
3. Compare the existing requirements with the new ones.
4. Remove those requirements that are not needed in the renovated system.
5. Make a new design of the desired system.
6. Code the new system.

General Reengineering Model

Eric J. Byrne suggests that reengineering is a sequence of three activities:

1. Reverse engineering
2. Re-design
3. Forward engineering



General Reengineering Model

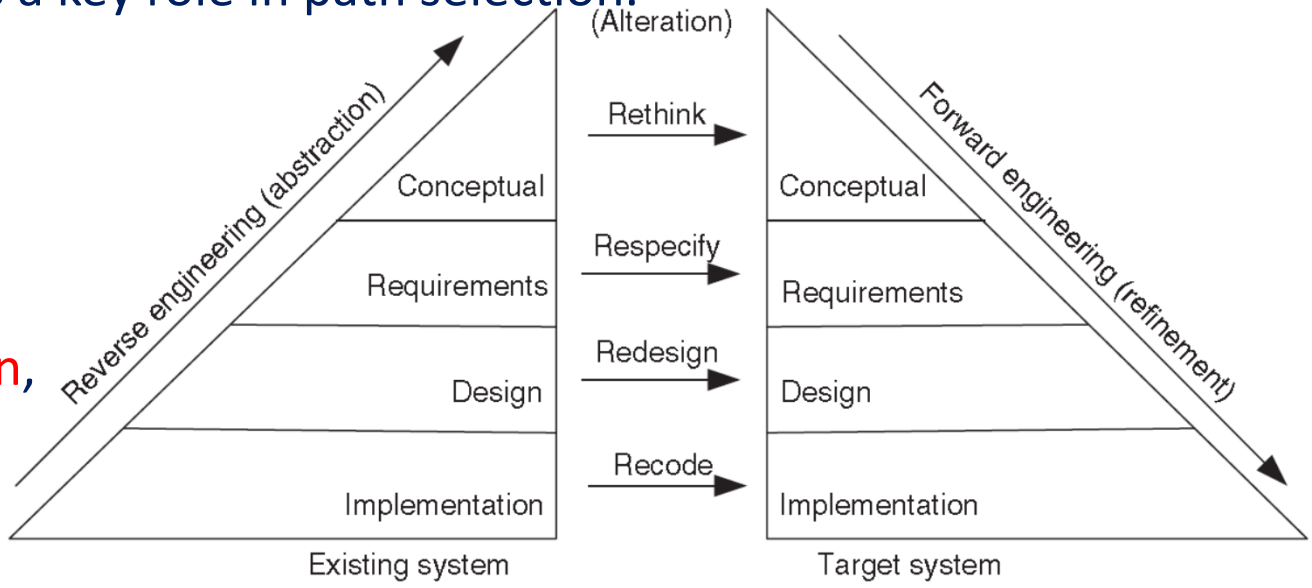
- Jacobson and Lindstorm define reengineering as:

  Reengineering = Reverse engineering + Δ + Forward engineering

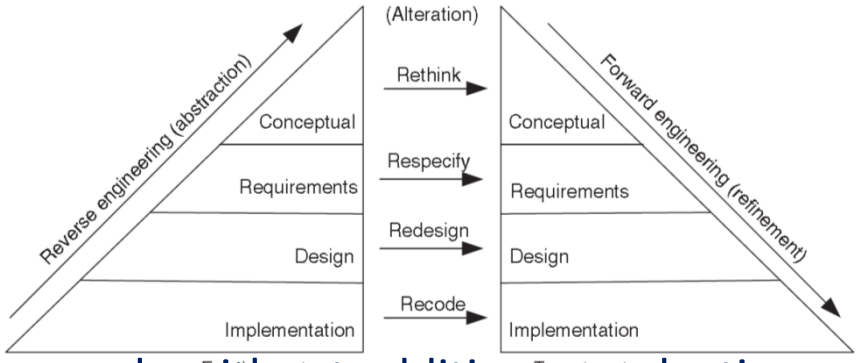
- The element "Δ" captures alterations made to the original system.
- Two major dimensions of alteration are: change in functionality and change in implementation technique.
  - A change in functionality comes from a change in the business rules.
  - Change of implementation technique. The end-user of a system never

Types of Changes

- The selection of a specific path for reengineering depends partly on the characteristics of the system to be modified.
- For a given characteristic to be altered, the abstraction level of the information about that characteristics plays a key role in path selection.
- Based on the type of changes required, reengineering characteristics are divided into groups: rethink, respecify, redesign, and recode

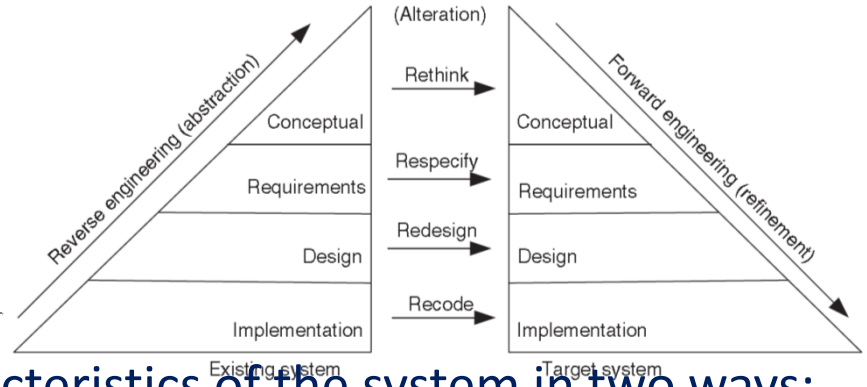


Types of Changes



- Recoding (also Rehosting) means reengineering of source code without addition or reduction of features in the transformed targeted source code.
  - Rehosting is most effective when the user is satisfied with the system's functionality, but looks for better qualities of the system.
  - Rephrasing and program translation are other means of source-code level changes
- Redesign refers to altering the design characteristics of the software by re-designing the system to :
  - Restructure the architecture
  - Modify the data model of the system
  - Replace a procedure or an algorithm with a more efficient one.

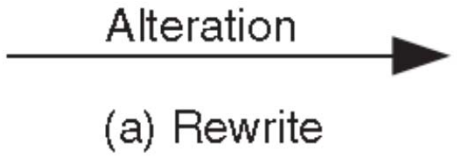
Types of Changes



- Respecify means changing the requirement characteristics of the system in two ways:
  - Change the form of the requirements.
  - Change the scope of the requirements.
- Rethink means manipulating the concepts embodied in an existing system to create a system that operates in a different problem domain.
  - Rethink involves changing the conceptual characteristics of the system, and it can lead to fundamental changes.
  - E.g. moving from the development of an ordinary cellular phone to the development of smartphone.

Software Reengineering Strategies

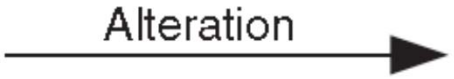
- Three strategies rewrite, rework, and replace specify the basic steps of reengineering
- Rewrite strategy: reflects the principle of alteration. By means of alteration, an operational system is transformed into a new system while preserving the abstraction level of the original system.
  - For example, the Fortran code of a system can be rewritten in the C language



(a) Rewrite

Software Reengineering Strategies

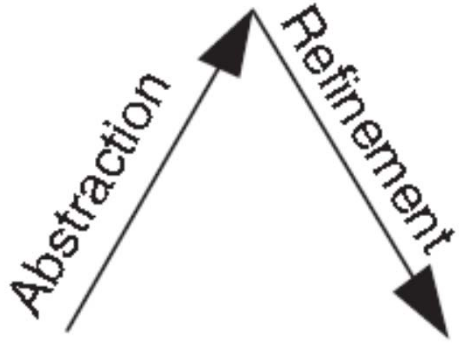
- Three strategies rewrite, rework, and replace specify the basic steps of reengineering
- Rewrite strategy: reflects the principle of alteration.



  - For example, the Fortran code of a system can be rewritten in the C language
- Rework strategy reflects the principles of abstraction , alteration and refinement .
  - For example, one can create an abstraction of source code in the form of a high-level design. Next, the reconstructed system model is transformed into the target system representation, by means of alteration. Finally, by means of refinement, an appropriate new system representation is created at a lower level of abstraction

Software Reengineering Strategies

- Replace strategy: applies abstraction and refinement
  - The system is reconstructed at a higher level of abstraction by hiding the details
  - a suitable representation for the target system is generated at a lower level of abstraction by applying refinement



Reengineering Process Variations

- Possible variations in reengineering processes can obtained from combing the reengineering strategies (rewrite, rework, and replace) and the change types (rethink, respecify, redesign, and recode)

Reengineering Approaches

- Five basic approaches to reengineering software systems exist:

1. Big Bang approach.
2. Incremental approach.
3. Partial approach.
4. Iterative approach.
5. Evolutionary approach:

- The five approaches are different in two aspects:

1. the extent of reengineering performed
2. the rate of substitution of the operational system with the new one.

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 disadvantage of Big Bang is that the reengineering project becomes a monolithic task, which may not be desirable in all situations.
- The Big Bang approach consumes too much resources at once for large systems, and takes a long stretch of time before the new system is visible.

Reengineering Approaches - Incremental Approach

- In the **Incremental Approach** 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:
  - locating errors becomes easier, because one can clearly identify the newly added components.
  - It becomes easy for the customer to notice progress, because of interim version release.
- The disadvantages of the incremental approach are as follows:
  - with multiple interim versions and their version controls, reengineering takes much longer time to complete
  - even if there is a need, the entire architecture of the system cannot be changed.

Reengineering Approaches- Partial Approach

- In the **partial approach** only a part of the system is reengineered and then it is integrated with the non-engineered portion of the system.
- The following three steps are followed in the partial approach:
  1. The existing system is partitioned into two parts: one part is identified to be reengineered and the remaining part to be not reengineered.
  2. Reengineering work is performed using either the “Big Bang” or the “Incremental” approach.
  3. The two parts the not-to-be-reengineered part and the reengineered part of the system, are integrated to make up the new system.
- The advantage is 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.

Reengineering Approaches- Iterative Approach

- The **iterative** 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.
- During the execution of the process four types of components can coexist:
  1. Old components not reengineered.
  2. Components currently being reengineered.
  3. Components already reengineered.
  4. New components added to the system.
- There are two advantages of the iterative reengineering process:
  - It guarantees the continued operation of the system during the execution of the reengineering process
  - The maintainers' and the users' familiarities with the system are preserved.

Reengineering Approaches - Evolutionary Approach

- In the **Evolutionary** approach components of the original system are substituted with re-engineered components.
- Software engineers focus their reengineering efforts on identifying functional objects irrespective of the locations of those components within the current system.
- There are two advantages of the Evolutionary approach:
  - The resulting design is more cohesive (built with functionally cohesive components).
  - The scope of individual components is reduced.
- A major disadvantage:
  - 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.

Reengineering Examples

Some well know reengineering examples are :

1. Code Reverse Engineering

2. The Source Code Reengineering Reference Model (SCORE/RM)

Code Reverse Engineering

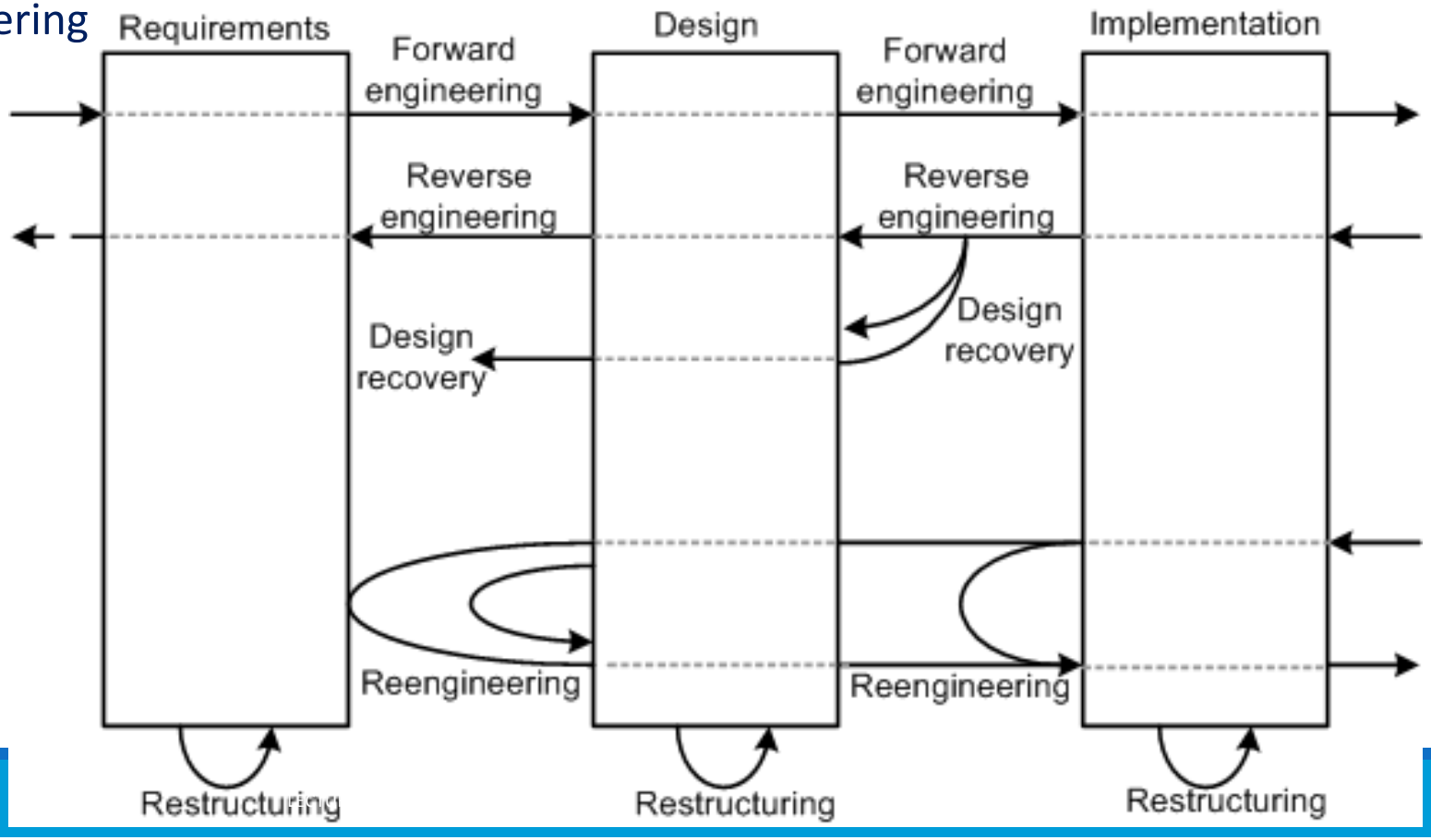
- Reverse engineering was first applied in electrical engineering to produce schematics from an electrical circuit.
- In the context of software engineering, Chikofsky and Cross define reverse engineering as a process to:
  - Identify the components of an operational software.
  - Identify the relationships among those components.
  - Represent the system at a higher level of abstraction or in another form.

Code Reverse Engineering

- Reverse engineering is performed to achieve two key objectives:
  - Redocumentation of artifacts
    - It aims at revising the current description of components or generating alternative views at the same abstraction level. Examples of redocumentation are pretty printing and drawing Control Flow Graphs (CFGs).
  - Design recovery
    - It creates design abstractions from code, expert knowledge, and existing documentation.

Code Reverse Engineering

Relationship between reengineering and reverse engineering



Code Reverse Engineering

- Six objectives of reverse engineering, as identified by Chikofsky and Cross II:
  1. Generating alternative views.
  2. Recovering lost information.
  3. Synthesizing higher levels of abstractions.
  4. Detecting side effects.
  5. Facilitating reuse.
  6. Coping with complexity.

Code Reverse Engineering

- Six key steps in reverse engineering, as documented in the IEEE Standard for Software Maintenance, are:
  1. Partition source code into units.
  2. Describe the meanings of those units and identify the functional units.
  3. Create the input and output schematics of the units identified before.
  4. Describe the connected units.
  5. Describe the system application.
  6. Create an internal structure of the system.
- The first three of the six steps involve local analysis, the rest involve global analysis.

Code Reverse Engineering

Reverse engineering has been effectively applied in the following problem areas:

- Redocumenting programs
- identifying reusable assets
- Discovering design architectures,
- building traceability between code and documentation
- Finding objects in procedural programs
- deriving conceptual data models
- Detecting duplications and clones
- cleaning up code smells
- Aspect-oriented software development
- Transforming binary code into source code
- Redesigning user interfaces
- parallelizing largely sequential programs
- Translating a program to another language
- Migrating data
- extracting business rules
- Wrapping legacy code
- Auditing security and vulnerability
- Extracting protocols of network applications

Techniques Used For Reverse Engineering

- The well-known analysis techniques that facilitate reverse engineering are:
  1. Lexical analysis.
  2. Syntactic analysis.
  3. Control flow analysis.
  4. Data flow analysis.
  5. Program slicing.
  6. Visualization.
  7. Program metrics.

Techniques Used For Reverse Engineering

1. Lexical Analysis

- Lexical analysis is the process of decomposing the sequence of characters in the source code into its constituent lexical units.
- A program performing lexical analysis is called a lexical analyzer, and it is a part of a programming language's compiler.
- Typically it uses rules describing lexical program structures that are expressed in a mathematical notation called regular expressions.
- Modern lexical analyzers are automatically built using tools called lexical analyzer generators, namely, lex and flex (fast lexical analyzer).

Techniques Used For Reverse Engineering

2. Syntactic analysis

- syntactic analyzers and parsers can be automatically constructed from a description of the programming language properties.
- Two types of representations are used to hold the results of syntactic analysis: parse tree and abstract syntax tree.
- An AST contains just those details that relate to the actual meaning of a program.
- Many tools have been based on the AST concept; to understand a program, an analyst makes a query in terms of the node types. The query is interpreted by a tree walker to deliver the requested information.

Techniques Used For Reverse Engineering

3. Control Flow Analysis (CFA)

- After determining the structure of a program, control flow analysis (CFA) can be performed on it.
- A CFG can directly be constructed from an AST by walking the tree to determine basic blocks and then connecting the blocks with control flow arcs.
- The two kinds of control flow analysis are:
  - Intraprocedural: It shows the order in which statements are executed within a subprogram.
  - Interprocedural: It shows the calling relationship among program units.

Techniques Used For Reverse Engineering

4. Data flow analysis (DFA)

- DFA concerns how values of defined variables flow through and are used in a program.
- CFA can detect the possibility of loops, whereas DFA can determine data flow anomalies.
  - One example of data flow anomaly is that an undefined variable is referenced.
- Data flow analysis enables the identification of code that can never execute, variables that might not be defined before they are used, and statements that might have to be altered when a bug is fixed.

Techniques Used For Reverse Engineering

5. Slicing

- A backward slice with respect to a variable v and a given point p comprises all instructions and predicates which affect the value of v at point p.
- Backward slices answer the question “What program components might effect a selected computation?”
- With respect to a variable v and a point p in a program, a forward slide comprises all the instructions and predicates which may depend on the value of v at p.
- Forward slicing answers the question “What program components might be effected by a selected computation?”

Techniques Used For Reverse Engineering

6. Visualization

- A software system is represented by means of a visual object to gain some insight into how the system has been structured.
- Two important notions of designing software visualization using 3D graphics and virtual reality technology are
  - Representation: This is the depiction of a single component by means of graphical and other media.
  - Visualization: It is a configuration of an interrelated set of individual representations related information making up a higher level component.

Techniques Used For Reverse Engineering

6. Visualization

- The following requirements are taken into account while designing a visualization:

1. Simple navigation.
2. High information content.
3. Low visual complexity.
4. Varying levels of detail.
5. Resilience to change.
6. Effective visual metaphors.
7. Friendly user interface.
8. Integration with other information sources.
9. Good use of interactions .
10. Suitability for automation.

Techniques Used For Reverse Engineering

7. Program Metrics

- Based on a module's fan-in and fan-out information flow characteristics, Henry and Kafura define a complexity metric,

  Cp = (fan-in × fan- out).

- A large fan-in and a large fan-out may be symptoms of a poor design.
- Six performance metrics are found in the Chidamber-Kemerer CK metric suite:

1. Weighted Methods per Class (WMC) – This is the number of methods implemented within a given class.
2. Response for a Class (RFC) – This is the number of methods that can potentially be executed in response to a message being received by an object of a given class.

Techniques Used For Reverse Engineering

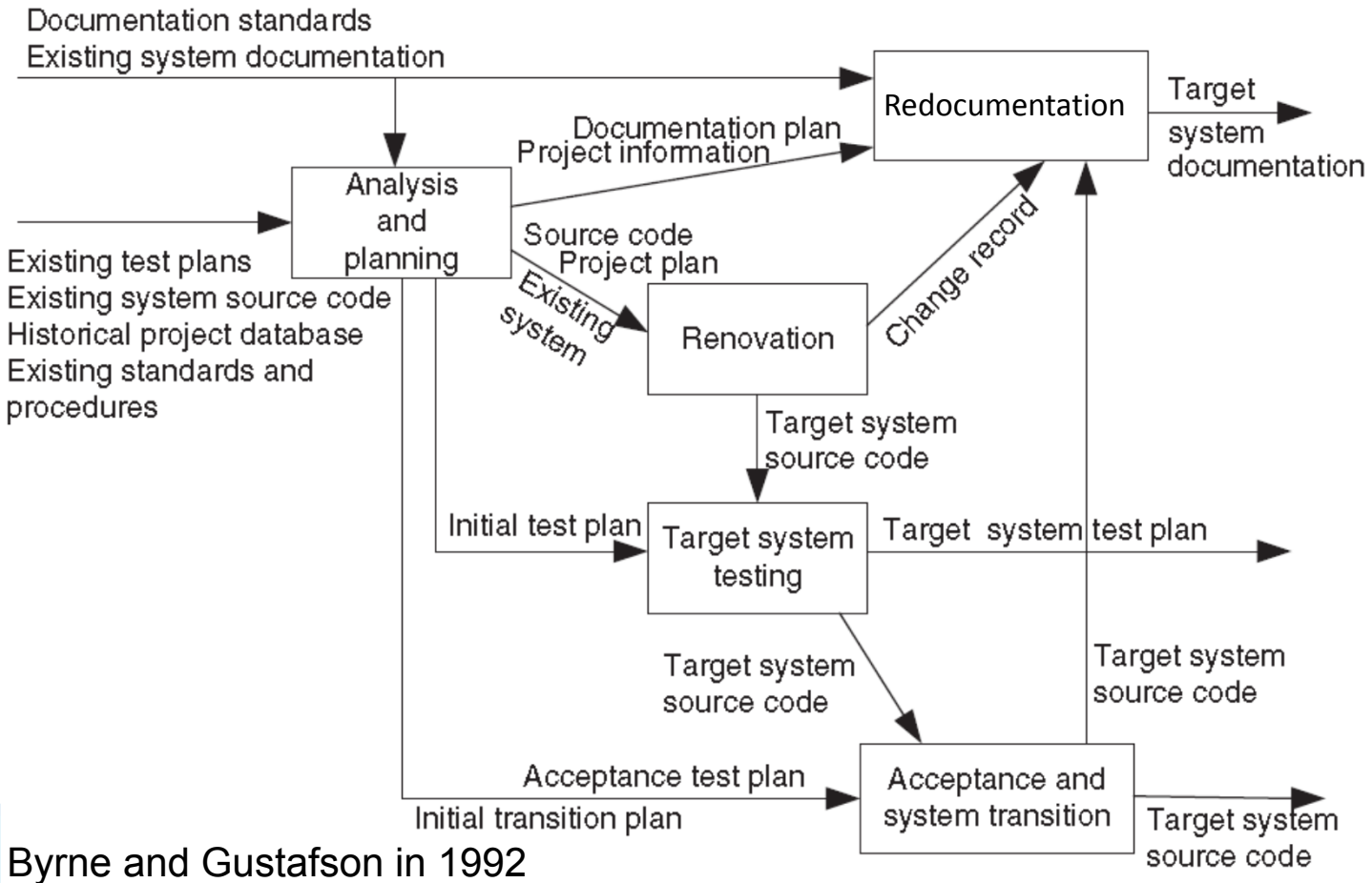
7. Program Metrics (cont'd)

3. Lack of Cohesion in Methods (LCOM) – For each attribute in a given class, calculate the percentage of the methods in the class using that attributes. Next, compute the average of all those percentages, and subtract the average from 100 percent.
4. Coupling between Object Class (CBO) – This is the number of distinct non-inheritance related classes on which a given class is coupled.
5. Depth of Inheritance Tree (DIT) – This is the length of the longest path from a given class to the root in the inheritance hierarchy.
6. Number of Children (NOC) – This is the number of classes that directly inherit from a given class.

The Source Code Reengineering Reference Model

- The model comprises five phases:

1. Analysis and Planning
2. Renovation
3. Target System Testing
4. Redocumentation
5. Acceptance Testing and System Transition

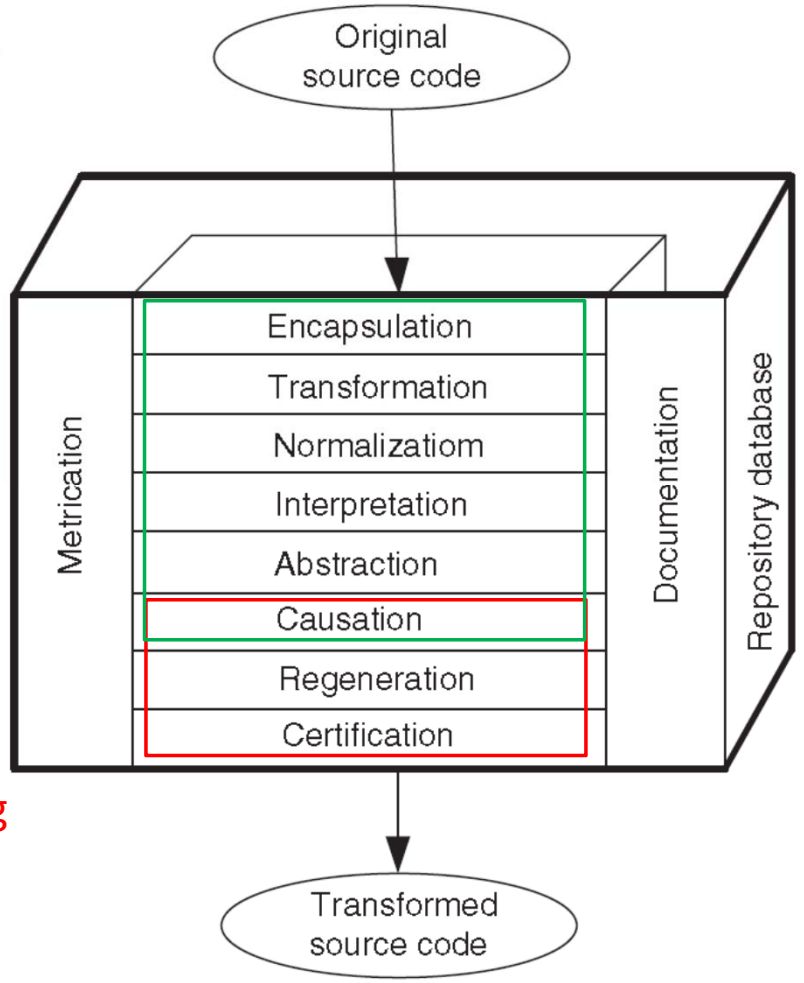


Byrne and Gustafson in 1992

The Source Code Reengineering Reference Model

- The Source Code Reengineering Reference Model (SCORE/RM) is useful in understanding the process of reengineering of software.
- The framework consists of four kinds of elements:

1. Function
2. Documentation
3. Repository Database
4. Metrication.



Reverse Engineering

Forward Engineering

The Source Code Reengineering Reference Model

1. Encapsulation:

This is the first layer of reverse engineering. In this layer, a reference baseline is created from the original source code. The goal of the reference baseline is to uniquely identify a version of a software and to facilitate its reengineering.

The following functions are expected of this layer:

- Configuration management.
- Analysis. The portions of the software requiring reengineering are evaluated.
- Test generation. This refers to the design of certification tests and their results for the original source code. Certification tests are basically acceptance tests

Data Reverse Engineering (DRE)

- Data Reverse Engineering (DRE) is defined as “the use of structured techniques to reconstitute the data assets of an existing system” .
- The purpose of DRE is as follows:

1. Knowledge acquisition.
2. Tentative requirements.
3. Documentation.
4. Integration.
5. Data administration.
6. Data conversion.
7. Software assessment.
8. Quality assessment.
9. Component reuse.

Database Reverse Engineering (DBRE)

- Database reverse engineering (DBRE) refers to recovering the specifications (i.e. the conceptual schema) in database of such applications
  - The conceptual schema is an abstract, implementation independent description of the stored data.
- A DBRE process facilitates understanding and redocumenting an application's database and files.
- By means of a DBRE process, one can recreate the complete logical and conceptual schemas of a database physical schema.
  - A logical schema describes the data structures in concrete forms as those are implemented by the data manager.
  - The physical schema of a database implements the logical schema by describing the physical constructs.

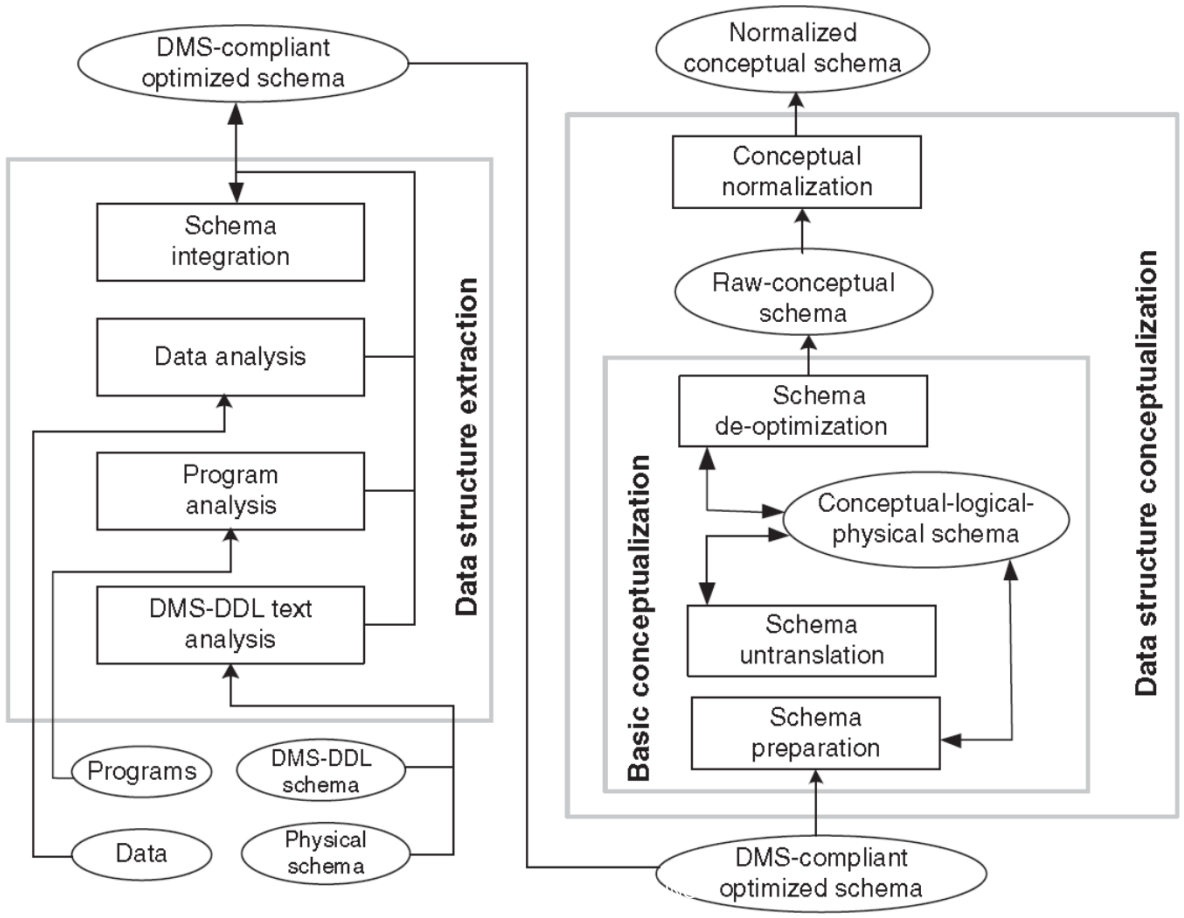
Database Reverse Engineering (DBRE)

- Database reverse engineering (DBRE) refers to recovering the specifications (i.e. the conceptual schema) in database of such applications
  - The conceptual schema is an abstract, implementation independent description of the stored data.
- A DBRE process facilitates understanding and redocumenting an application's database and files.
- By means of a DBRE process, one can recreate the complete logical and conceptual schemas of a database physical schema.
  - A logical schema describes the data structures in concrete forms as those are implemented by the data manager.
  - The physical schema of a database implements the logical schema by describing the physical constructs.

Database Reverse Engineering (DBRE)

- The forward design process of a database comprises three basic phases as follows:
  1. Conceptual phase: The user requirements are gathered, studied, and formalized into a conceptual schema.
  2. Logical phase: the conceptual schema is expressed as a simple model, which is suitable for optimization reasoning (ER or ORM).
  3. Physical phase: the logical schema is described in the data description language (DDL) of the data management system and the host programming language.
- A DBRE process is based on backward execution of the logical phase and the physical phase.
- The process is divided into two main phases:
  1. data structure extraction.
  2. data structure conceptualization.

Database Reverse Engineering (DBRE)



Questions

?