Software Re-Engineering

Lecture: 10

Dr. Imran Ali
School of Computing- Software Engineering
FAST-National University of Computer & Emerging Sciences, Karachi

Sequence [Todays Agenda]

Content of Lecture

- Code Reverse Engineering

Code Reverse Engineering

- Reverse engineering was first applied in electrical engineering to produce diagrams from an electrical circuit.
- It was defined as the process of developing a set of specifications for a complex hardware system by an orderly examination of specimens of that system.

Code Reverse Engineering

- In the context of software engineering, Chikofsky and Cross II defined reverse engineering as a process to:

(i) Identify the components of an operational software;

(ii) Identify the relationships among those components;

(iii) Represent the system at a higher level of abstraction or in another form.

- In other words, by means of reverse engineering one derives information from the existing software artifacts and transforms it into abstract models to be easily understood by maintenance personnel.

Factors of Reverse Engineering

- The factors necessitating the need for reverse engineering are as follows:
  - The original programmers have left the organization.
  - The language of implementation has become obsolete, and the system needs to be migrated to a newer one.
  - There is insufficient documentation of the system.
  - The business relies on software, which many cannot understand.
  - The company acquired the system as part of a larger acquisition and lacks access to all the source code.
  - The system requires adaptations and/or enhancements.
  - The software does not operate as expected.

Factors of Reverse Engineering

- These discussed factors imply that a combination of both high-level and low-level reverse engineering steps need to be applied.

High-level Reverse Engineering

- High-level reverse engineering focuses on understanding the functionality and design of a system or product without needing to examine every detail of its implementation, often using tools like netlist analysis to reconstruct high-level control logic.
- This approach is particularly useful in situations where understanding the overall structure and behavior is more important than the specifics of the code.
- Netlist:
  - It is a file format that describes the components, connections, and sometimes placement and routing. It's essentially a list of all the components and how they're connected to each other.

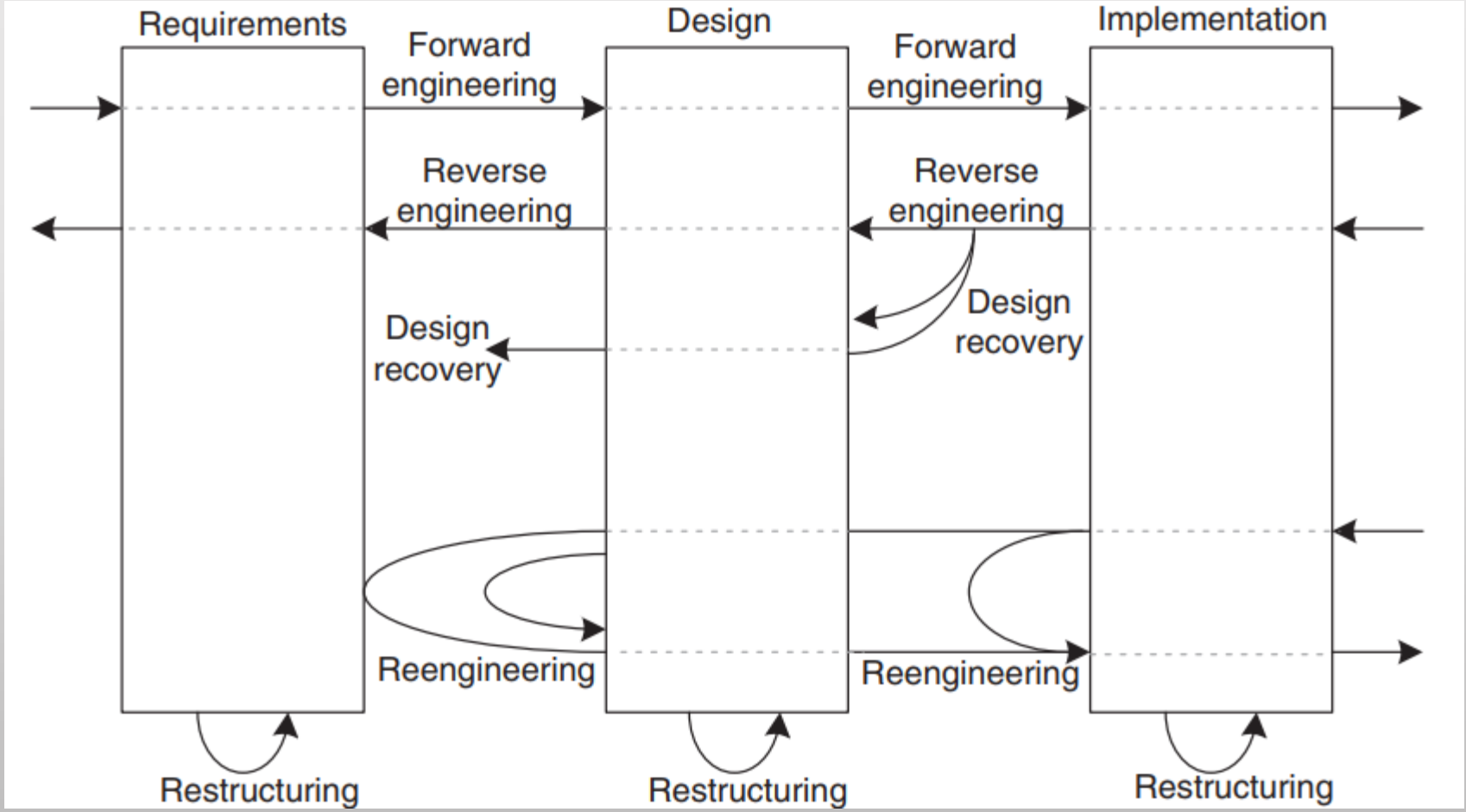
Low-level Reverse Engineering

- It means creating source code from object code or assembly code.

- 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 CFGs.
- Design recovery:
- It creates design abstractions from code, expert knowledge, and existing documentation.
- In design recovery the domain knowledge, external information, and deduction or fuzzy reasoning are added to the observations of the subject system.

- Pretty Printing:
- Pretty printing is the process of formatting data, especially source code, to make it visually appealing and easy to read. It involves applying stylistic conventions like indentation, color-coding, and spacing to highlight syntactic elements. While pretty printing can be applied to various data structures, it's particularly relevant in the context of Context-Free Grammars (CFGs).
- A CFG defines the syntax of a language.

- The relationship between forward engineering, reengineering, and reverse engineering is shown in below Figure.



Objectives of Reverse Engineering

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

Objectives of Reverse Engineering

- Six key steps in reverse engineering, as documented in the IEEE Standard for Software Maintenance, are:
  - Partition source code into units.
  - Describe the meanings of those units and identify the functional units.
  - Create the input and output schematics of the units identified before.
  - Describe the connected units.
  - Describe the system application.
  - Create an internal structure of the system.

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

- redocumenting programs
- identifying reusable assets
- discovering design architectures
- recovering design patterns
- 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
- computing change impact
- 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

- A high level organizational paradigm is found to be useful while setting up a reverse engineering process, as advocated by Benedusi et al. The high level paradigm plays two roles:
- Define a framework to use the available methods and tools, and
- Allow the process to be repetitive.
- The paradigm, namely, Goals/Models/Tools, which partitions a process for reverse engineering into three ordered stages: Goals, Models, and Tools.

- Goals:
- In this phase, the reasons for setting up a process for reverse engineering are identified and analyzed.
- Analyses are performed to identify the information needs of the process and the abstractions to be created by the process.
- The team setting up the process first acquires a good understanding of the forward engineering activities and the environment where the products of the reverse engineering process will be used.
- Results of the aforementioned comprehension are used to accurately identify:
  - The information to be generated.
  - The formalisms to be used to represent the information

- Models:
- In this phase, the abstractions identified in the Goals stage are analyzed to create representation models.
- Representation models include information required for the generation of abstractions.
- Activities in this phase are:
  - Identify the kinds of documents to be generated.
  - To produce those documents, identify the information and their relations to be derived from source code.
  - Define the models to be used to represent the information and their relationships extracted from source code.
  - To produce the desired documents from those models, define the abstraction algorithm for reverse engineering.

- Tools:
- In this phase, tools needed for reverse engineering are identified, acquired, and/or developed in-house.
- Those tools are grouped into two categories:
  - Tools to extract information and generate program representations according to the identified models.
  - Tools to extract information and produce the required documents.
  - Extraction tools generally work on source code to reconstruct design documents.
- Therefore, those tools are ineffective in producing inputs for an abstraction process aiming to produce high-level design documents.


Thank You!