SWE-417 SOFTWARE REENGINEERING

Software Engineering Department

Sir Syed University of Engineering & Technology

Week No. 7

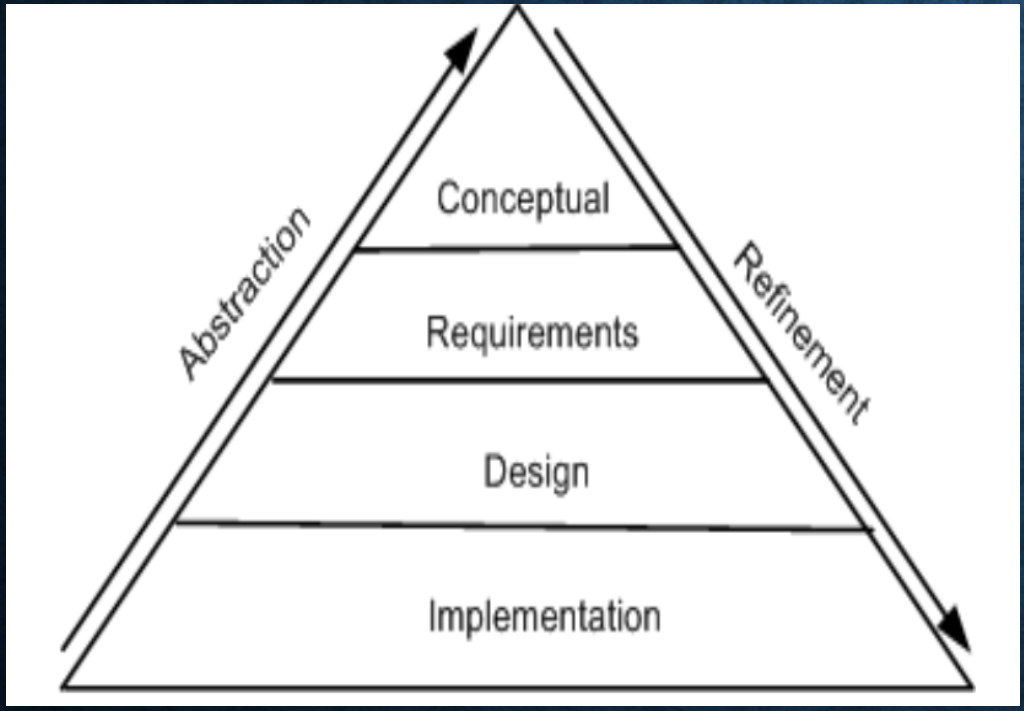
CONTENTS

- RE-Engineering Techniques
  - Overview of Reverse engineering, Restructuring & Forward engineering

RE-ENGINEERING

- Re-engineering, also known as reverse engineering or software re-engineering, is the process of analyzing, designing, and modifying existing software systems to improve their quality, performance, and maintainability.
- This can include updating the software to work with new hardware or software platforms, adding new features, or improving the software's overall design and architecture.
- Software systems are reengineered 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

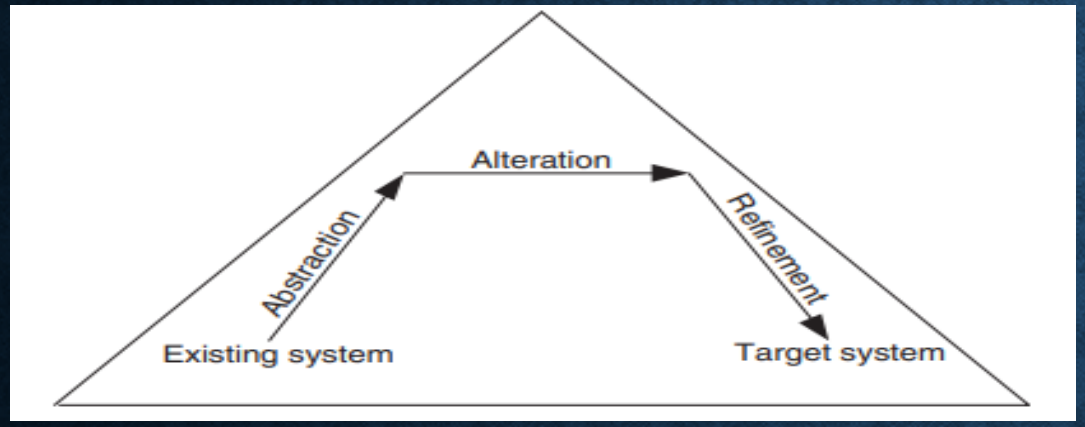
RE-ENGINEERING

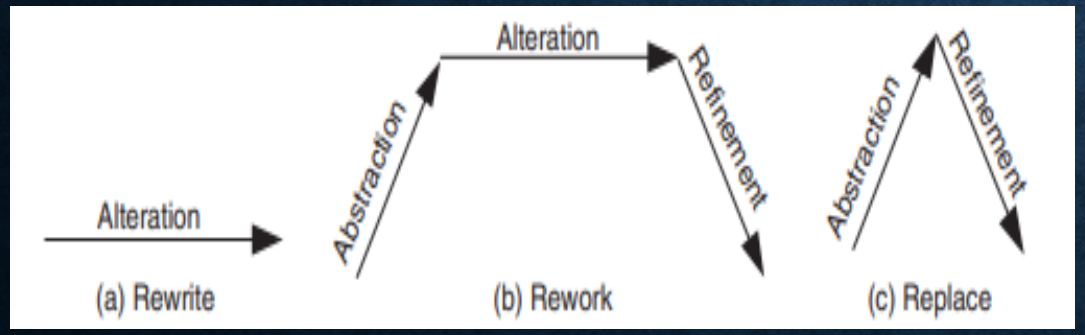


"Levels of abstraction and refinement"

- Conceptual level: Why does the system exist?
- Requirements level: What does the system do?
- Design level: (i) "What are the characteristics of the system?" (ii) "How is the system going to possess the characteristics to deliver the functionalities?
- Implementation level: "how" exactly the system is implemented.

RE-ENGINEERING

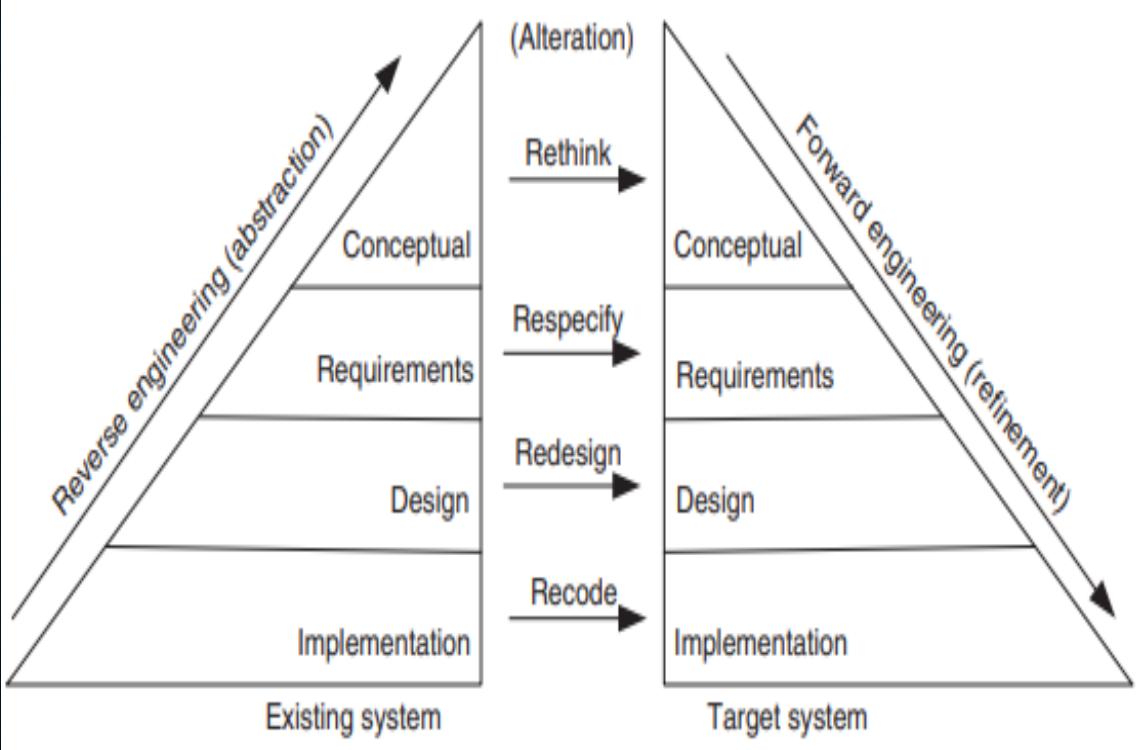




"Conceptual basis for the reengineering process"

- Abstraction: system can be gradually increased by successively replacing the details with abstract information.
- Refinement: system is gradually decreased by successively replacing some aspects of the system with more details.
- Alteration: making of some changes to a system representation is known as alteration.

GENERAL MODEL FOR SOFTWARE REENGINEERING



"General model of software reengineering"

- The reengineering process accepts as input the existing code of a system and produces the code of the renovated system.
- For example, a program written in C++ can be translated into a new program in PYTHON.
- This model 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, respectively.

A GENERAL MODEL FOR SOFTWARE REENGINEERING

- 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.

REVERSE ENGINEERING

- Reverse engineering is the process of analyzing a system, product, or software to understand its components, functionality, and design. This involves dissecting and studying the system without access to its original design documentation or source code.
- Systematic evaluation of a product with the purpose of replication.
  - Design of a new part
  - Copy of an existing part
  - Recovery of a damage or broken part

WHY REVERSE ENGINEERING

- Reverse engineering serves various purposes across different fields, industries, and contexts. Some common reasons why reverse engineering is employed:
  - **Understanding Existing Systems:** Reverse engineering allows individuals or organizations to understand how existing systems, products, or technologies work. This is particularly useful when dealing with legacy systems or when trying to integrate new technologies with existing ones.
  - **Product Improvement:** Engineers often use reverse engineering to analyze competitors' products and identify areas for improvement.
  - **Legacy System Maintenance:** As technologies evolve, companies may find themselves relying on legacy systems that are no longer supported or well-documented. Reverse engineering can be employed to understand these systems, fix bugs, or make necessary updates.

IMPORTANCE

- We are not start from the very beginning to develop a new product every time.
- We need to optimize the resources available in our hands and reduce the production time keeping in view the customers' requirement.
- **Scenario for Reverse Engineering**: Imagine a company has a legacy software application critical to its operations, but the original source code and documentation are no longer available. The software is outdated, and the company is facing compatibility issues with modern systems. They decide to perform reverse engineering to understand the existing software and make necessary updates.

- **Steps in Software Reverse Engineering:**
  - **Initial Analysis:**
    - Identify the main functionalities of the software.
    - Determine the software's inputs, outputs, and overall behavior.
  - **Decompilation:**
    - Use decompilers to analyze the compiled code and attempt to recreate a high-level source code representation. Understand the algorithms and logic used in the original software.
  - **Static and dynamic analysis:**
    - Both static and dynamic analysis are used in tandem to gain a comprehensive understanding of a software system. This combined approach provides a more thorough assessment of the software's structure, behavior, and potential issues.

- **Documentation:**
  - Create documentation based on the insights gained from reverse engineering. Document the software's architecture, data flow, and key components.
- **Code Reconstruction:**
  - Using the information gathered, reconstruct a high-level source code representation of the software. Implement any necessary updates or improvements.
- **Testing and Validation:**
  - Test the reconstructed software to ensure that the modifications did not introduce errors. Validate that the updated software functions as intended and addresses compatibility issues.

CODE REVERSE ENGINEERING

- ***Code Reverse Engineering*** is the process of analyzing software to understand its structure, functionality, and behavior, often without having access to the original documentation or design details.
- Goals of Code Reverse Engineering:
  - Understand Legacy Systems: To maintain or update old software where documentation is missing or incomplete.
  - Extract Design Insights: To learn the architectural and logical flow for better documentation or optimization.
  - Enhance Security: To identify vulnerabilities, ensure compliance, or detect malware.
  - Rebuild Compatibility: To adapt software for newer platforms or environments.

CODE REVERSE ENGINEERING

- Challenges
  - Obfuscated Code: Deliberately complex or protected code can be difficult to reverse-engineer.
  - Legal and Ethical Boundaries: Reverse engineering may violate intellectual property laws or software licenses in some jurisdictions.
  - Complex Systems: Analyzing large, poorly documented, or legacy systems can be time-consuming and resource-intensive.

CODE 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.
- 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.
- The first three of the six steps involve local analysis, the rest involve global analysis.

CODE REVERSE ENGINEERING

- Software applications comprise **source code** files that are compiled to convert them into binary executable code.
- If this binary executable code is converted back into **source code** files using a decompiler then this will be termed as **reverse engineering of source code** or code Reverse Engineering.

Reverse Engineering Requirements:

- First, we need **source code** files of the software such as C/C++ code files and/or java files of the software etc.

CODE REVERSE ENGINEERING

- If the **source code** files of the software are not available then we need to decompile the software, Decompilation is the process of converting machine code (binary) back into a higher-level programming language, or at least a representation that is closer to the original source code.

Tool: Decompilers like IDA Pro, Hopper can be used to analyze and decompile binaries.

- If the purpose of **reverse engineering** was to explore the **source code** then we need a **source code** editor for searching texts, functionalities, libraries, and/or algorithms in the decompiled **source code** files and/or shared **source code files**.

Tool: I**ntegrated Development Environment (IDE) Cloud storage platform (**For example, GitHub repository, GitLab etc.)

- If the purpose of **reverse engineering** was to make changes in the **source code** files then using the source code editor we can add and/or delete code modules and/or functionalities.

Tool: One popular and widely used example of a source code editor is **Visual Studio Code (VSCode).**
It is a free and open-source code editor developed by Microsoft.

QUESTION

Scenario: Intelligent Chatbot with Sentiment Analysis

Develop a chatbot capable of understanding and responding to user queries while analyzing the user's sentiment (positive, negative, neutral) and adjusting its tone accordingly. Apply six key steps of reverse engineering to an Intelligent Chatbot with Sentiment Analysis project:

SOLUTION

- Step 1: Partition Source Code into Units

The chatbot's source code is divided into logical units based on its core components. For the intelligent chatbot, the identified units might include:

1. Input Handling: Preprocessing user input.
2. Sentiment Analysis: Determining the sentiment (positive, negative, neutral) of the input.
3. Intent Recognition: Understanding the user's intent or query.
4. Response Generation: Crafting an appropriate response based on intent and sentiment.
5. Learning Engine: Optimizing responses using reinforcement learning.

- Tools:
  - Code editors: **Visual Studio Code** or **JetBrains PyCharm**: For exploring and refactoring code.
  - Code Analysis Tools: **SonarQube** : For analyzing source code and identifying logical units.
  - Version Control: **Git** : For managing and tracking code changes.

SOLUTION

- Step 2: Describe the Meanings of Those Units and Identify the Functional Units. Each unit is analyzed to understand its functionality:

1. Input Handling:
    - Cleans and preprocesses user input, such as tokenization and removing noise (e.g., stopwords, emojis). Converts input to a machine-readable format.
2. Sentiment Analysis:
    - Uses a pre-trained model (e.g., BERT or GPT) to classify the user's sentiment as positive, negative, or neutral.
3. Intent Recognition:
    - Identifies the purpose of the query (e.g., asking a question, making a complaint, or seeking guidance).

SOLUTION

4. Response Generation:
    - Produces a response tailored to the user's sentiment and intent, ensuring tone adjustment.
5. Learning Engine:
    - Uses reinforcement learning to improve conversational flow by adjusting future responses based on feedback.

- Tools:
    - Code Documentation Tools: **Doxygen**: For generating detailed documentation directly from the source code. **Sphinx**: For creating technical documentation for Python-based projects.

SOLUTION

- Step 3: Create the Input and Output Schematics of the Units Identified Before. Each unit's inputs and outputs are mapped:

1. Input Handling:
   - Input: Raw user text.
   - Output: Preprocessed text for analysis.
2. Sentiment Analysis:
   - Input: Preprocessed text.
   - Output: Sentiment score/classification (positive, negative, neutral).
3. Intent Recognition:
   - Input: Preprocessed text.
   - Output: Identified intent (e.g., complaint, query, feedback).

SOLUTION

4. Response Generation:
   - Input: Sentiment classification and identified intent.
   - Output: Contextual response (e.g., “I’m sorry to hear that” for negative sentiment).
6. Learning Engine:
   - Input: User feedback, interaction logs.
   - Output: Optimized response patterns.

- Tools:
  - Data Flow Diagram (DFD) Tools: **Lucidchart, Draw.io, or Microsoft Visio**: For creating input/output schematics and data flow diagrams.
  - Testing Frameworks: **Postman**: For testing input-output behavior of APIs. **Pytest**: For unit testing Python modules.

SOLUTION

- Step 4: Describe the Connected Units, the connections between units are described:
  - Input Handling feeds data into both Sentiment Analysis and Intent Recognition units.
  - Sentiment Analysis and Intent Recognition share their outputs with Response Generation.
  - Learning Engine uses feedback from all units to optimize their performance.
- Tools:
  - Dependency Graph Tools: **Graphviz**: For visualizing connections between units.
    PlantUML: To model relationships among the components.

SOLUTION

- Step 5: Describe the System Application

The Intelligent Chatbot with Sentiment Analysis is designed to assist users by understanding their queries and adjusting its tone based on their emotional state. It can serve as:

  - Customer Support Agent: Addressing customer queries and complaints.
  - Mental Health Assistant: Offering empathetic responses and guiding users toward resources.
  - Educational Assistant: Helping learners by answering questions in an encouraging manner.

- Tools:
  - Project Management Platforms: **Jira, Asana, or Trello**: To document the chatbot's application and track feature development.
  - Reporting and Documentation: **Microsoft Word or Google Docs**: To create comprehensive documentation of system applications.

SOLUTION

- Step 6: Create an Internal Structure of the System. An internal structural diagram (e.g., UML Component Diagram) can be created, showing:
  - Modules: Input Handling ,Sentiment Analysis , Intent Recognition & Learning Engine
  - Data Stores: Pre-trained Models (e.g., BERT, GPT) , Conversation Logs & Reinforcement Learning Feedback
  - Interactions: Data flows between modules. Connections to external APIs (e.g., sentiment analysis models or databases).

Outcome:

By applying these steps, the engineering team fully understands the structure, function, and flow of the Intelligent Chatbot with Sentiment Analysis, enabling them to document the system comprehensively and identify areas for improvement or modification.

RESTRUCTURING

- Restructuring is a process of rearranging or reconstructing the source code and decide whether to retain or change the programming language.
- Software restructuring is a form of perfective maintenance that modifies the structure of a program's source code.
- The objective of restructuring is to eliminate or restructure the source code parts that may cause performance issues.
- Apart from this, removing obsolete code or system parts is also an essential process.

RESTRUCTURING

- In restructuring , the parts of the source code then often cause errors in the software can be changed or updated.
- In restructuring, we eliminate older version of certain parts of the system (like programming implementation and hardware components) so that software will perform up to-date.

FORWARD ENGINEERING

- Forward engineering is same as software engineering process, but it is carried out after reverse engineering in Re-engineering.
- Forward engineering is the traditional process of designing and creating a system, product, or software from scratch. In this approach, designers and developers start with requirements and proceed to create the system using standard development practices.
- Forward Engineering applies of all the software engineering process which contains SDLC to recreate associate existing application. It is near to full fill new needs of the users into re-engineering.

SUMMARY

- Reverse Engineering, analyzes existing systems to understand their design, functionality, and architecture.
- Restructuring, involves improving the internal structure of code or systems without changing functionality.
- Forward Engineering, builds or upgrades systems from abstract designs or conceptual models.