Reengineering Concepts

Idaho State University | Computer Science

Isaac Griffith

CS 4423 and CS 5523
Department of Computer Science
Idaho State University

ROAR

Outcomes

After today's lecture you will:

- Understand the basic concepts of Reengineering
- Understand a generalized model for Reengineering
- Understand the primary strategies of Reengineering

Reengineering

CS 4423/5523

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.
- The goal of reengineering is to:
  - understand the existing software system artifacts, namely, specification, design, implementation, and documentation
  - improve the functionality and quality attributes of the system
- 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

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:
    1. focusing on the more significant information about the system
    2. Hiding the irrelevant details at the moment
- On the other hand, refinement is the reverse of abstraction

Reenginering Concepts

- **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.

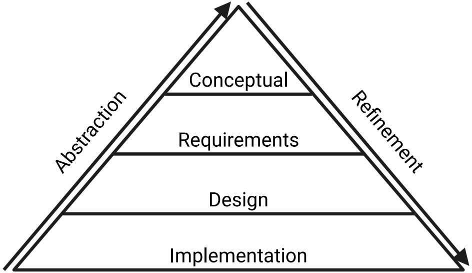
Reengineering Concepts

- A new software is created by going downward from the top, highest level of abstraction to the bottom, lowest level. This downward movement is know as **forward engineering**.
- **Forward engineering** follows a sequence of activities; formulating concepts about the system to identifying requirements to designing the system to implementing the design.
- On the other hand, the upward movement through the layers of abstractions is called **reverse engineering**
- **Reverse engineering** of software systems is a process comprising the following steps:
  1. analyze the software to determine its components and the relationships among the components
  2. represent the system at a higher level of abstraction or in another form
- **Decompilation** is an example of Reverse Engineering, in which object code is translated into a high-level program

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**

- The four levels are:
  - Conceptual
  - Requirements
  - Design
  - Implementation

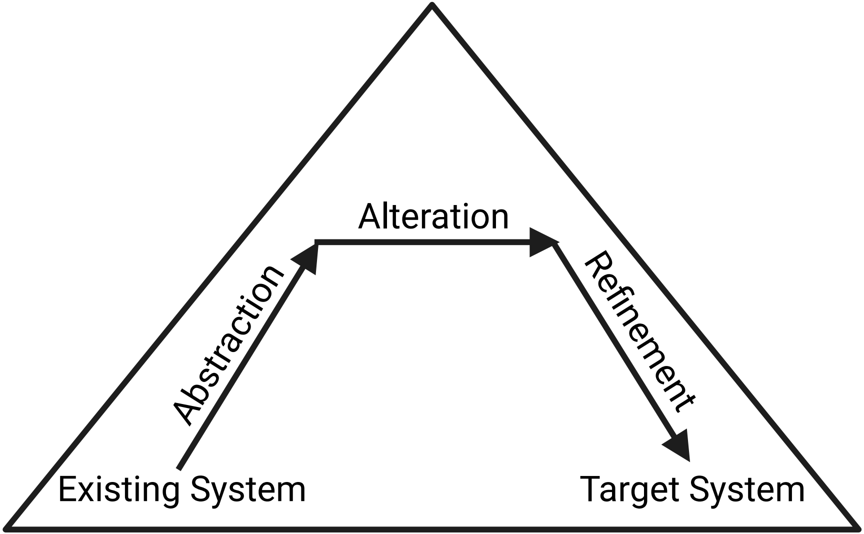


- The refinement process:
  - **why? → what? → what & how? → how?**
- The abstraction process:
  - **how? → what & how? → what? → why?**

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 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

Reengineering Concepts

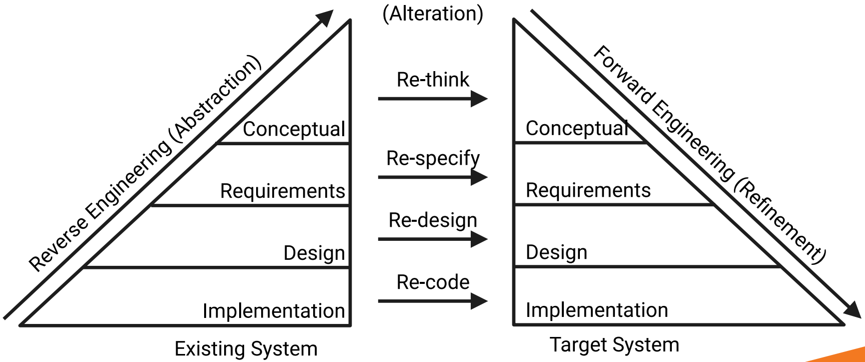


A General Model for Reengineering

- 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

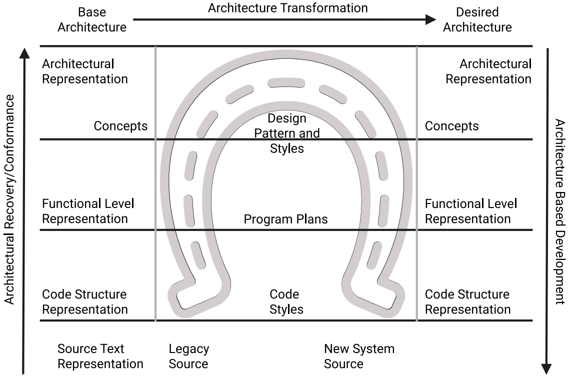
A General Model for Reengineering

- The model 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



A General Model for Reengineering

- A visual metaphor, called horseshoe, 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


Base Architecture
Architecture Transformation
Desired Architecture
Architectural Representation
Architectural Representation
Concepts
Design Pattern and Styles
Concepts
Functional Level Representation
Program Plans
Functional Level Representation
Code Structure Representation
Code Styles
Code Structure Representation
Source Text Representation
Legacy Source
New System Source
Architectural Recovery/Conformance
Architecture Based Development

A General Model for Reengineering

Now, we are in a position to re-visit three definitions of reengineering

- **Chikofsky and Cross II:** Software reengineering is the analysis and alteration of an operational system to represent it in a new form and to obtain a new implementation from the new form. Here, a new form means a representation at a higher level of abstraction
- **Byrne:** Reengineering of a software system is a process for creating a new software from an existing software so that the new system is better than the original system in some ways.
- **Arnold:** Reengineering of a software system is an activity that:
  1. improves the comprehension of the software system
  2. raises the quality levels of the software, namely, performance, reusability, and maintainability

A General Model for Reengineering

In summary, it is evident that reengineering entails:

❶ the creation of a more abstract view of the system by means of some reverse engineering activities

❷ the restructuring of the abstract view

❸ implementation of the system in a new form by means of forward engineering activities

A General Model for Reengineering

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

$$Reengineering = ReverseEngineering + \Delta + ForwardEngineering$$

- The 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

General Model for Reengineering

- Another common term used by practitioners of reengineering is **rehosting**
- **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
- Examples of better qualities are improved efficiency of execution and reduced maintenance costs

Types of Change

Based on the types of changes required, system characteristics are divided into groups:

- **rethink**
- **respecify**
- **redesign**
- **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 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:
  1. restructuring and architecture
  2. modifying the data model of the system
  3. replacing a procedure or an algorithm with a more efficient one

Respecify

- This means changing the requirement characteristics of the system in two ways:
  1. change the form of the requirements
  2. change the scope of the requirements

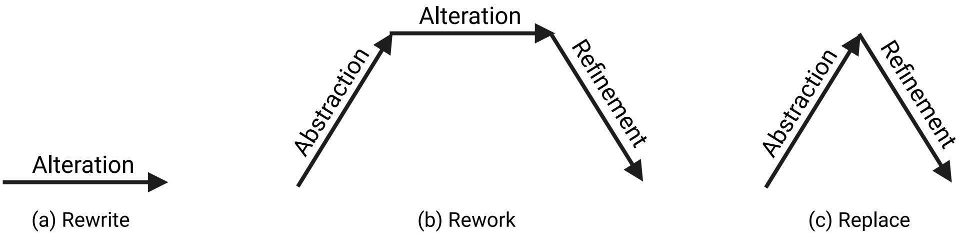
Rethink

- 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

Software Reengineering Strategies

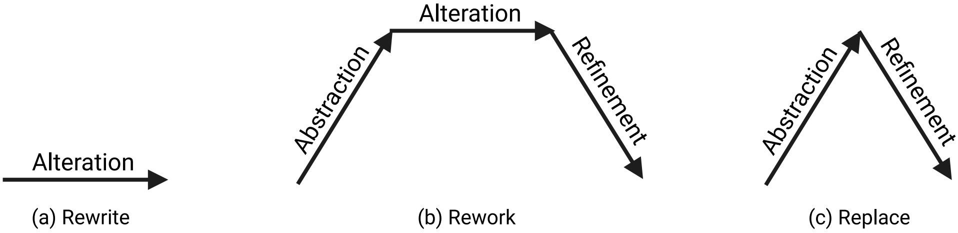
Three strategies that specify the basic steps of reengineering are:

1. **rewrite**
2. **rework**
3. **replace**



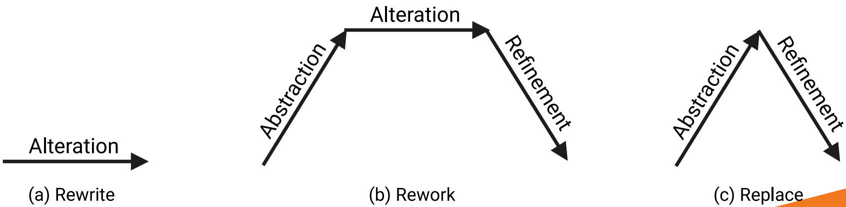
Rewrite Strategy

- This strategy reflects the principle of alternation.
- 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



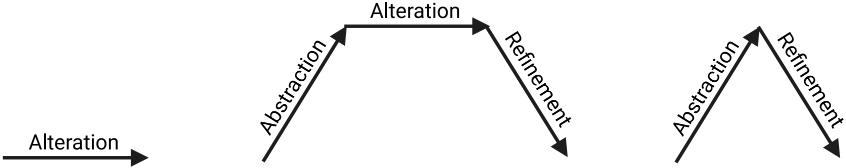
Rework Strategy

- The rework strategy applies all three principles
- If 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
- Then, 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



Replace Strategy

- The replace strategy applies two principles, namely, abstraction and refinement.
- To change a certain characteristic of a system:
  1. the system is reconstructed at a higher level of abstraction by hiding the details of the characteristic
  2. 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



(a) Rewrite (b) Rework (c) Replace

Reengineering Variations

Starting Abstraction Level	Type of Change	Rewrite	Rework	Replace
Implementation Level	Re-code	Yes	Yes	Yes
	Re-design	Bad	Yes	Yes
	Re-specify	Bad	Yes	Yes
	Re-think	Bad	Yes*	Yes*
Design Level	Re-code	No	No	No
	Re-design	Yes	Yes	Yes
	Re-specify	Bad	Yes	Yes
	Re-think	Bad	Yes*	Yes*
Requirement Level	Re-code	No	No	No
	Re-design	No	No	No
	Re-specify	Yes	Yes	Yes
	Re-think	Bad	Yes*	Yes*
Conceptual Level	Re-code	No	No	No
	Re-design	No	No	No
	Re-specify	No	No	No
	Re-think	Yes	Yes*	Yes*

For Next Time

- Review EVO Chapter 4.1 - 4.3
- Read EVO Chapter 4.4 - 4.6
- Watch Lecture 08

Are there any questions?