Advanced Topics in Software Engineering

Unit 4

Red: indicates important

Contents

- Formal methods
- Cleanroom software engineering
- Software reuse
- Component based software engineering
- Software reengineering

Formal Methods

- Formal method is that area of computer science that is concerned with the application of mathematical techniques to the design and implementation of computer software.
- Two essential components are:
  1. Formal language : A formal language is a set of strings over some well-defined alphabet.
  2. Formal reasoning : Formal methods support formal reasoning about formulae in the language. These are exemplified by formal proofs. A proof begins as a set of axioms, which are taken as statements postulated to be true. Inference rules state that if certain formula, known as premises, are derivable from the axioms, then another formula, known as the consequent, is also derivable . A set of interference rules must be given in each formal method. A proof consist of a sequence of well-defined formulae in the language in which each formula is either an axiom or derivable by an interference rule from previous formulae in the sequence. The last formula in the sequence is said to be proven.

- Main parts to formal method :
  1. Formal specification : Using mathematics to specify the desired properties of a computer system.
  2. Formal verification : Using mathematics to prove that a computer system satisfies its specification.
  3. Automated programming : Automating the process of program generation.
- Two major concerns of formal method :
  1. The correctness of the problem
     - Producing software that is correct is difficult
     - By using rigorous mathematical techniques, it may be possible to make probably correct software.
  2. Programs are mathematical objects
     - They are expressed in a formal language;
     - They have a formal semantics;
     - Programs can be treated as mathematical theories

- Well-known formal methods :
  - Z (pronunciation “Zed”)
  - Communicating Sequential Processes (CSP)
  - Vienna Development Method
  - Larch
  - Formal Development Methodology(FDM)
- What can be formally specified?
- Formal methods can include graphical languages. Data Flow Diagrams (DFDs) are the most well-known technique for specifying the function of a system. DFDs are semi-formal method.
- Petri nets provide another well-known graphical technique. Its used in distributed system. Petri nets are fully formal technique.

- Well-known formal methods :
  - Z (pronunciation "Zed")
  - Communicating Sequential Processes (CSP)
  - Vienna Development Method
  - Larch
  - Formal Development Methodology(FDM)
- What can be formally specified?
- Formal methods can include graphical languages. Data Flow Diagrams (DFDs) are the most well-known technique for specifying the function of a system. DFDs are semi-formal method.
- Petri nets provide another well-known graphical technique. Its used in distributed system. Petri nets are fully formal technique.
- System engineers frequently use formal methods. Hardware engineers also use formal methods, such as VHSIC Hardware Description Language (VHDL) descriptions, to model integrated circuits before fabricating them.

Goals of formal specifications

- Removal of ambiguity : The formal syntax of specification language enables requirements or design to be interpreted in only one way. This eliminates the ambiguity that occurs with natural language. It also encourages greater rigour in the early stages of the software engineering process.
- Consistency : Facts stated in one place should not be contradicted. This is ensured by mathematically proving that the initial facts can be formally mapped using interfaces into later statements later in the specification.
- Completeness : This is difficult while using formal method. Some aspects of the system may be left undefined by mistake or by intention. It is not possible to consider every operational scenario in a complex system.

Application Areas of formal specifications

- Formal methods can be used to specify aspects of a system other than functionality.
- Its also used for software safety and security.
- Its used in research organization in areas like fault tolerance, response time, space efficiency, reliability, human factors and software structure dependencies.
- Other application areas are :
  - Safety critical systems
  - Security systems
  - The definition of standards
  - Hardware development
  - Operating systems
  - Transaction processing systems
  - Anything that is hard, complex, or critical.

Limitations of formal specification using formal method

- It mainly focuses on functions and data. Its difficult to represent on timing, control and behavioral aspects of a problem.
- Some of the elements like HCI, human-machine interface) are better specified using graphical techniques. It is difficult to learn using a specification using formal method than other
- Some of the drawbacks are :
  - Cost : Its as costly as to do a fully formal specification as to do all the coding. The only solution for this is not to formally specify everything.
  - Complexity : Everybody cannot read formal specification. The solution for this is to provide a very good and detailed documentation. It should have explanations in natural language.
  - Deficiencies of less formal approach : The methods of structured and object oriented design make use of natural languages and a variety of graphical notations. Careful application of these analysis and deign methods leads to high quality software. Sloppiness in the application of these methods lead to following problems :
    - Contradictions : statements that are at variance with one another.
    - Ambiguities : statements that can be interpreted in a number of ways.

Limitations of formal specification using formal method

- Vagueness : statements with less precision, and contribute little information.
- Incomplete statements : a description is not functionally complete.
- Mixed levels of abstractions : statements with high and low levels of detail are interspersed, which can make it difficult to comprehend the functional architecture.

Cleanroom Software Engineering

- Was developed by Dr. Harlan Mills.
- Its an engineering and managerial process for the development of high-quality software with certified reliability.
- Its emphasis is on defect prevention rather than defect removal.
- Cleanroom software engineering gives software that:
  - Is correct by mathematically sound design
  - Is certified by statistically-valid testing
  - Reduces the cycle time resulting from an incremental development strategy and avoidance of rework.
  - Is well-documented.
  - Detects errors as early as possible and reduces the cost of errors during development and the incidence of failures during operations.

- Following principles are the foundation for the Cleanroom-based Software development:

1. Incremental development under statistical quality control (SQC) :
   - It provides a basis for statistical quality control of the development process.
   - Each increment is a complete iteration of the process, and measures of performance in each increment are compared to pre-established standards to determine whether the process is "in control". If quality standards are not met, testing of the increment ceases and developers return to design phase.
2. Software development based on mathematical principles:
   - In cleanroom software engineering, the key principle is that a computer program is an expression of a mathematical function.
   - The Box Structure Method is used for specification and design, and functional verification is used to confirm that the design is a correct implementation of the specification.
   - The specification must define that function before design and functional verification can begin.
   - Verification is performed through correctness is performed through team review based on correctness questions.

3. Software testing based on statistical principles :
   - In cleanroom , software testing is considered as a statistical experiment.
   - A subset of all possible uses of the software is generated, and performance of the subset is used as a basis for operational performance.

Conventional Software Engineering Vs Cleanroom Software Engineering

- These are the areas where both are different :
- Small team of software engineers :

❖ A cleanroom project team is small, usually 6 to 8 members.

❖ It works in an organized way to ensure the intellectual control of work in progress.

❖ The cleanroom process teams comprise of :

1. Specification team : Develops and maintains the system specification.
2. Development team : Develops and verifies software
3. Certification team : develops set of statistical tests to exercise software after development and reliability growth models used to assess reliability.

Conventional Software Engineering Vs Cleanroom Software Engineering

- Time allocation across the life cycle phases:

❖ Major objective of cleanroom is to prevent errors from occurring, the amount of time spent in the design .

❖ So, the amount of time spent in the design phase of cleanroom engineering is greater than the amount of time spent in design phase of conventional software engineering.

- Existing organizational practices :

❖ Cleanroom software engineering does not prevent using other software engineering techniques as long as they are compatible with cleanroom principles.

- Examples of projects implemented in cleanroom software engineering:
- IBM COBOL/SF product
- Ericson OS-32 operating system project
- USAF space Command and Control Architectural Infrastructure(SCAI)
- US Army Cleanroom project in the Tank-automotive and Armaments Command at the U.S.Army Picatinny Arsenal.

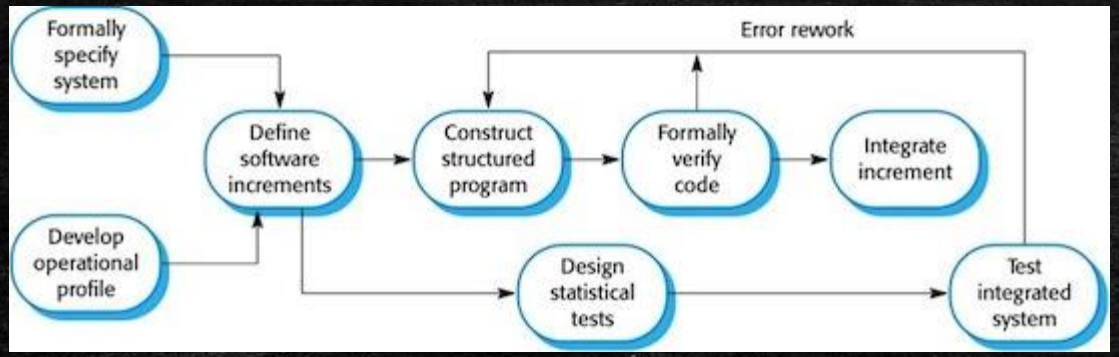
Cleanroom Software Engineering Principles, Strategy and Process Overview

- The software development is based on mathematical principles. It follows the box principle for specification and design. It uses formal verification to confirm correctness of implementation. Program correctness is verified by team reviews using questionnaires. Testing is based on statistical principles,

- The following phases take place in Cleanroom Software Engineering:
  1. Incremental planning : The project plan is built around the incremental strategy.
  2. Requirements gathering: Customer requirements are elicited and refined for each increment using traditional methods.
  3. Box structure specification: The box structure separate and isolate the behavior, data and procedure in each increment.
  4. Formal design : Specifications are iteratively refined to become architectural designs and component-level designs. Architectural designs are called as state-boxes and component-level designs are called as clear boxes.
  5. Correctness verification: Correctness questions are asked and answered, and formal mathematical verification is used as required.

Cleanroom Software Engineering Principles, Strategy and Process Overview

6. Code generation, inspection and verification: Box structures are translated into program language, inspections are done to ensure conformance of codes and boxes, syntactic correctness of code.
7. Statistical test planning: A suit of test cases is created to match the probability distribution of the projected product usage pattern.
8. Statistical use testing: Statistical sample of all possible test cases is used rather than exhaustive testing.
9. Certification: After the verification, inspection and correctness of all errors, the increments are certified as ready for integration.



Overview of Cleanroom Software Engineering Development

Limitations Of Cleanroom Engineering

- They are too theoretical, too mathematical and too radical for use in real software development.
- Relies on correctness verification and statistical quality control rather than unit testing
- Organizations operating at the ad hoc level of the CMM do not make rigorous use of the defined processes needed in all phases of the software lifecycle.

Similarities Between Cleanroom and OO Paradigm

- Lifecycle: both rely on incremental development
- Usage: Cleanroom usage model similar to OO use case
- State Machine Use: cleanroom state box and OO transition diagram
- Reuse: Explicit objective in both process models.

Difference between Cleanroom and OO

Cleanroom	OO
Relies on decomposition	Relies on composition
Relies on formal methods	Allows informal use case definition and testing
Cleanroom usage hierarchy is a system itself	OO inheritance hierarchy is a design resource
Cleanroom practitioner's prefer tabular representation	OO practitioner's prefer graphical representation
Tool support is there only for testing and not design.	Tool support is good for most OO processes

Software Reuse and its types

- It's the process of creating software systems from existing software components.
- In most engineering disciplines, systems are designed by composing existing components that have been used in other systems.
- To achieve better software, more quickly and at lower cost, we need to adopt a design process based on systematic software reuse.
- Some of the components that can be reuse are as follows;
  - Source code
  - Design and interfaces
  - User manuals
  - Software Documentation
  - Software requirement specifications and many more.

- Types of software reuse:

- Application System reuse: an application may be reused either by incorporating one application inside another(COTS reuse) or by developing application families.(e.g.. MS Office)

- Component reuse: Components of an application reused in another application.

- Function reuse: reusing software components that implement a single well-defined function.

- Advantages of software reusability:
  - Increase software productivity
  - Shorten software development time
  - Reduced process risk : Less uncertainty in development costs
  - Increased reliability: components already exercised in working systems.
  - Effective use of components : Reuse components instead of people.
  - Standards compliance : Embed standards in reusable components.
  - Accelerated development : Avoid custom development and speed up delivery.
  - Reduce software development and maintenance cost

Component Based Software Engineering(CBSE)- Why CBSE?

- Goal of CBSE is to increase productivity, quality, and decrease time-to-market in software development.
- Software system is built from standard components rather than ' reinventing the wheel' each time.
- It focuses on reusing and adapting existing components.
- The process begins when a software team establish requirements for a system to be built using conventional requirements techniques.
- An architectural design is established, but rather than moving immediately into more designed tasks, the team examines requirements to determine what subset is directly amenable to composition, rather than construction.

What Is A Component? What are its types?

- It's a nontrivial, Independent, and replaceable parts of a system that fulfills a clear function in the context of a well-defined architecture.
- They have an interface.
- They employ inheritance rules.
- It can be characterized based on its use in the CBSE process. One category is Commercial off-the-shelf (COTS) components. These are components that can be purchased, pre-built, with the disadvantages that is no source code available, and so the definition of the use of the component given by the manufacturer, and the services which it offers, must be thoroughly tested, as they may or may not be accurate. Advantage is that these components are more robust and adaptable as they are used and tested in many different applications.

What Is A Component? What are its types?

- It's a nontrivial, Independent, and replaceable parts of a system that fulfills a clear function in the context of a well-defined architecture.
- They have an interface.
- They employ inheritance rules.
- It can be characterized based on its use in the CBSE process. One category is Commercial off-the-shelf (COTS) components. These are components that can be purchased, pre-built, with the disadvantages that is no source code available, and so the definition of the use of the component given by the manufacturer, and the services which it offers, must be thoroughly tested, as they may or may not be accurate. Advantage is that these components are more robust and adaptable as they are used and tested in many different applications.
- In addition to COTS, CBSE also provides:
  - Qualified components
  - Adapted components
  - Assembled components
  - Updated components

- For each requirement, we must question:
  - Are Commercial Off-The-Shelf(COTS) components available to implement the requirements?
  - Are internally developed reusable components available to implement the requirements?
  - Are the interfaces for available components compatible within the architecture of the system to be built?
  - The CBSE process is characterized in a manner that not only identifies candidate components but also qualifies each component's interface, adapts components to remove architectural mismatches, assembles components into selected architectural style, and updates components as requirements for the system change.
- Two processes occur in parallel during the CBSE process. They are:
  - Domain Engineering
  - Component Based Development

Domain Engineering

- The intent of domain engineering is to identify, construct, catalogue, and disseminate a set of software components that have applicability to existing and future software in a particular application domain.
- Paul Clements, defined domain engineering as a process of finding commonalities among systems to identify components that can be applied to many systems and to identify program families that are positioned to take fullest advantage of those components.
- Examples of application domain:
  - Air traffic control system
  - Defence systems
  - Financial market systems

- It begins by identifying the domain to be analysed. This is achieved by analyzing existing applications and by consulting experts of the type of application to be built.
- A domain model is then realized by identifying operations and relationships that recur across the domain and therefore becomes candidates for reuse.
- One particular approach to domain engineering is Structural Modeling. It's a approach that works under the assumption that every application domain has repeating patterns that has reuse potential.
- There are 3 stages in this process:
  1. Qualification
  2. Adaption
  3. Composition

- Qualification:
- It examines reusable components.
- It ensures that a candidate component will perform the function required, will properly fit into the architectural style specified for system, and will exhibit quality characteristics that are required for application.
- Interface does not provide all the information required to determine if a proposed component an be reused effectively in the new application.
- The three important characteristics considered are: performance, reliability and usability

- Adaption:
- Also known as wrapping
- Even after a component has been qualified for use within an application architecture, conflicts may occur in one or more of the areas. To avoid these conflicts an adaption technique called wrapping is often used.
- Some common approaches for this are:
- White-box wrapping: Integration conflicts are removed by making code-level modifications to the code. This is unlikely in the case of COTS as they do not have source code.
- Grey-box wrapping: used when component library provides a component extension language or API that allows conflicts to be removed or masked.
- Black-box wrapping: Its used hen source-code is not available. Requires the introduction of pre- and post-processing the interface level to remove conflicts.

- Composition:

- The component composition task assembles qualified, adapted and engineered components to populate the architecture established for an application. To accomplish this, an infrastructure must be established to bind the components into an operating system.

- There are many mechanisms for creating an effective infrastructure in order to achieve component composition. Some of theses are:
  1. Data Exchange Model: allows users and applications to interact and transfer data. Example: drag-and-drop, cut-and-paste. It should be defined for all reusable components
  2. Underlying object model: ensures that components developed in different languages are interoperable across computing platforms.

Component Technologies Available

- CORBA: allows components to communicate remotely and transparently over network.
- Java Beans and Microsoft COM: allows components from different vendors to work together within Windows.

Advantages of CBSE

- Reduces development time
- Increases productivity
- Reduction in cost for development
- Reliability is increased
- Management of complexity
- Flexibility

Challenges of CBSE

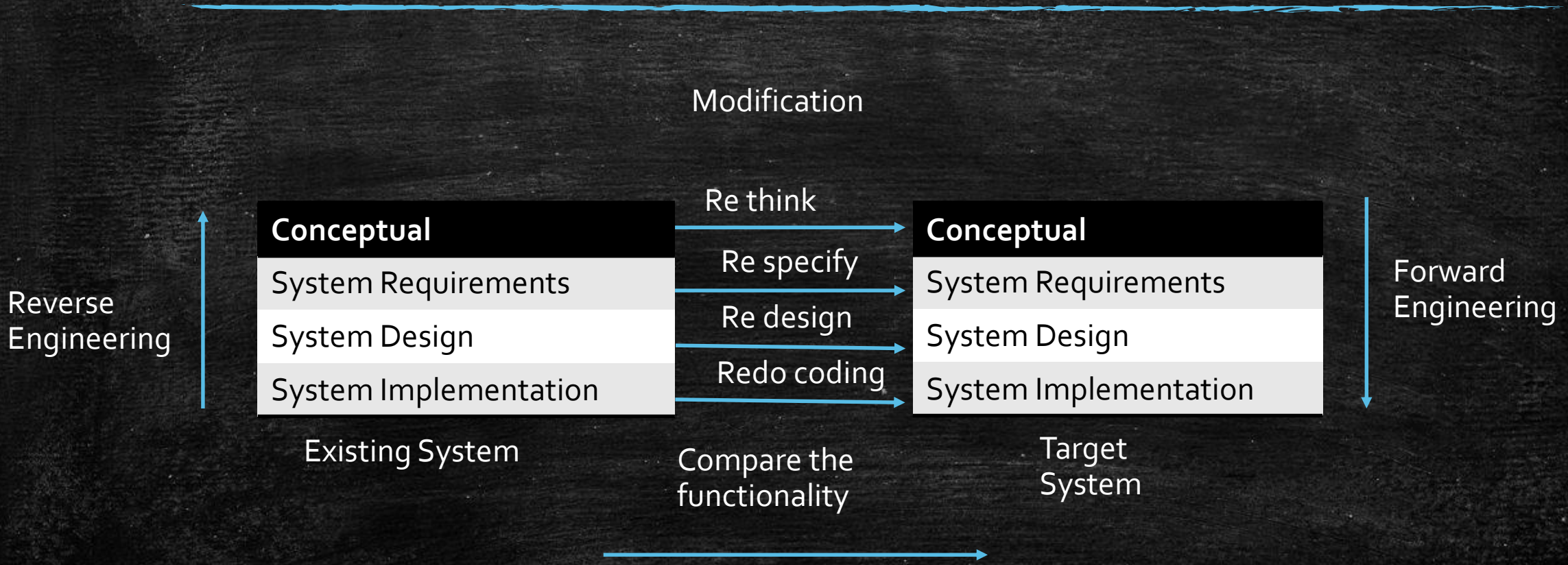
- Dependable systems and CBSE: The use of CBD in safety-critical domains, real-time systems, and different process -control systems in which the reliability requirements are more rigorous which is challenging. A major problem with CBD is the limited possibility of ensuring the quality and other non-functional attributes of the components.
- Tool support: The use of software engineering is to provide solutions to practical problems, and the existence of appropriate tools is essential for a successful CBSE. There are successful development tools such as Visual Basic. There are many other tools yet to appear such as component selection and evaluation tools, run-time analysis tools, component test tools, component configuration tools etc..
- Trusted Components: trend is to deliver the components in binary form and the component development process is not in control of component users, so trustworthiness is of great importance.
- Component Certification: Certification gives the results of tests performed and description of the environment in which the tests is performed. Certification is not yet established in software and software components.

- Component predictability: We can specify the relevant attributes of the components, it is not known how these attributes can determine the corresponding attributes of the system.
- Requirements management and component selection: Requirements management is a complex process. A problem of requirements management is incomplete, imprecise and contradictory. There are many uncertainties in component selection, there is a need for a strategy for managing risks in the component selection and evolution process.
- Long-term management of component based system: As component based system include sub-systems and components, the problem of system evolution becomes significantly more complex. CBSE is a new approach so has little experience and is difficult to maintain.
- Development models: Existing development models demonstrate powerful technologies, they have many unambiguous characteristics, they are incomplete and they are difficult to use.

- Component configuration: Complex systems include many components which in turn include other components. As we work with complex structures, the problems with structure configuration arise.

Software Reengineering

- It's a process of software development which is done to improve the quality of a software system.
- It is the examination, analysis, and alteration of an existing software system to reconstitute it in a new form.
- It encompasses a combination of sub-processes such as reverse engineering, re-documentation, restructuring, and forward engineering.
- The goal is to understand the existing software system components and to improve performance, functionality or implementation.
- In reengineering, reverse engineering is done to existing code to extract design and requirements. Forward reengineering is then done to develop the replacement system.



- Reasons to reengineer legacy system(old system):
  - Allow legacy system to quickly adapt to changing requirements.
  - Comply to new organizational standards.
  - Upgrade to new technologies/platforms/paradigm
  - Extend the software's life expectancy
  - Identify candidates for reuse
  - Improve software maintainability
  - Increase productivity
  - Reduce maintenance error and costs.

Objectives of Reengineering

- Objectives of software reengineering depends on the goals of the client and users of the system.
- Two general objectives of reengineering are:
  - Improve quality: The existing software system is of low quality, due to modifications and out-of-date documentation. Re-engineering is intended to improve the software quality and to produce current documentation. Improved quality is needed to increase reliability, to improve maintainability, to reduce the cost of maintenance.
  - Migration: Old working software may still meet user's needs, but it may depend on hardware platforms, operating systems or languages that have become obsolete and thus must be re-engineered, migrating to a newer platform or language.

Software Reengineering Life Cycle

- Different phases in software reengineering are:
  - Requirements analysis phase
  - Model analysis phase
  - Source code analysis phase
  - Remediation phase
  - Transformation phase
  - Evaluation phase

Requirements Analysis:

- It refers to identification of concrete reengineering goals for a given software.
- Specifications of the criteria should be specified and illustrated in the new reengineered process.
- Violations that need to be repaired must also be identified .

Software Reengineering Life Cycle

Model analysis phase: It refers to documenting and understanding the architecture and functionality of the legacy software. In order to understand a legacy system, its necessary to capture its design, its architecture and the relationship between different elements of its implementation. Reverse engineering the legacy system is done to extract design information from its code.

Source code analysis phase: it refers to the identification of the parts of the code that are responsible for violation of requirements specified in the analysis phase. It encompasses the design of methods and tools to inspect, measure , rank and visualize software structures. Problem detection can be static analysis or dynamic analysis. Static analysis is done by analyzing the source code or its design structure. Dynamic analysis is investigation of how grogram behaves at run time.

Remediation phase: It refers too the selection of a target software structure that aims to repair a design or a source code defect. Since legacy system have evolved from classes, objects, and methods in such a way that they heavily depend on each other, a detected problem must be divided into sub-problems.

Software Reengineering Life Cycle

Transformation phase: it consists of physically transforming software structures according to the remediation strategy selected. This requires methods and tools to manipulate and edit software systems, re-organize and re-compile them automatically, debug and check their consistency, and manage different versions of the software being reengineered.

Evaluation phase: It refer to the process of assessing the new system as well as establishing and integrating the revised system throughout the corporate operating environment. It needs training and possibly the need for adopting a new improved business process model.