Software Re-Engineering

Lecture: 01

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

Self Introduction

BE- Sukkur IBA University
MS- Sukkur IBA University
PhD- SNNU, Xi'an, China

Research Interests:

- Software Engineering
- Granular Computing
- Fuzzy Sets & Systems
- Data Analysis
- Information Processing
- Granulation Measures
- Formal Concept Analysis

Course Information

Course Instructor

Dr. Imran Ali
Email: imran.ali@nu.edu.pk

Office Hours

Main Campus: Monday - Thursday 08:00 AM – 04:00 PM

CODE OF ETHICS

- All the students must come to class on time

  (Attendance will be taken on regular basis)
- Students should remain attentive during class and avoid use of **Mobile phone**, **Laptops** or any **gadgets**
- Respect faculty and staff through actions and speech
- Class participation is encouraged

Course Introduction

- **Course Objectives:**
  - This course helps students to understand and practice different software reengineering techniques.
  - The participants of this course will learn how to apply reengineering techniques to maintain and modify software systems.
  - **Prerequisites:**
  - Software Construction and Development (SE-3001)
  - **Mode**
  - Mix of lectures, demonstrations, discussions, exams

Text Book

- **Primary**
  - **Re-Engineering legacy software**, David Lorge Parnas, Chris Birchall, Safari Books, Shelter Island, NY, 2016.
  - S. Demeyer, S. Ducasse, and O. Nierstrasz. **Object-Oriented Reengineering Patterns**. Morgan Kaufmann, 2002.

- **Reference**
  - **Application Software Reengineering**, Afshar alam, Tendai Patenga, Dorling Kindersley Pvt. Ltd, Pearson Education in South Asia, 2010.
  - **Software Maintenance and Evolution: a Roadmap**, K.H.Bennett and V.T Rajlich, The Future of Software Engineering, ACM Press 2000.

Course Outline

Week #	Class Topics
Week 1	Introduction to Software Re-Engineering
Week 2	Software Evolution
Week 3	Legacy Systems
Week 4	RE-Engineering Techniques (reverse engineering, restructuring & forward propagation)
Week 5	Reverse Engineering & its techniques
Week 6	Refactoring code to analysis artifacts
Week 7	Refactoring code to architecture
Week 8	Object Oriented Re-engineering Patterns

Course Outline

Week #	Class Topics
Week 9	Object Oriented Re-engineering Patterns cont.
Week 10	Object Oriented Re-engineering Patterns cont.
Week 11	Code restructuring
Week 12	Program comprehension
Week 13	Quality issues in re-engineering processes
Week 14	Tool support for Re-engineering, Challenges & Stakeholder aspiration
Week 15	Software maintenance and re-engineering economics.
Week 16	Student Presentation

Course Learning Outcome

- At the end of this course, the students should be able to:
  - **CLO-1: Define** the concepts and techniques of software reengineering.
  - **CLO-1: Demonstrate** and utilize reengineering techniques to maintain and modify software systems.
  - **CLO-2: Examine** maintenance related problems associated with object-oriented software systems
  - **CLO-2: Investigate** and design complex reengineering and reverse engineering problems

Evaluation Components

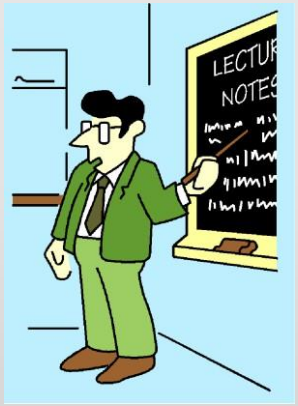
- Quizzes - 6
- Assignments - 6
- Project - 8
- Midterm - 30
- Final - 50
- Project (in group of max 3 students)
- Demonstration

COURSE INFORMATION

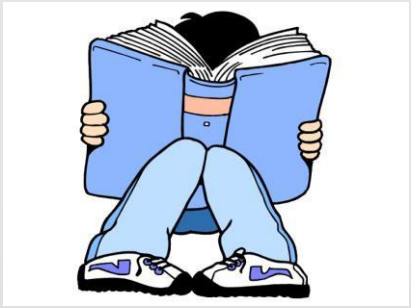
Google Classroom:

GCR Code BSE-8A: bxghqzj
GCR Code BSE-8B: s7c7rsw

Class Composition

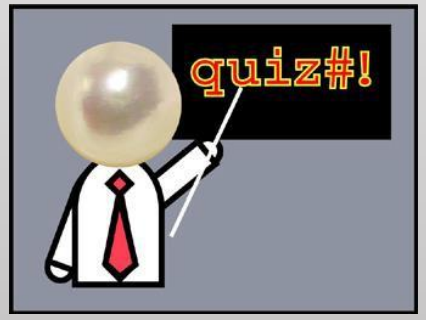


Lectures



Readings

Term Project



Quizzes

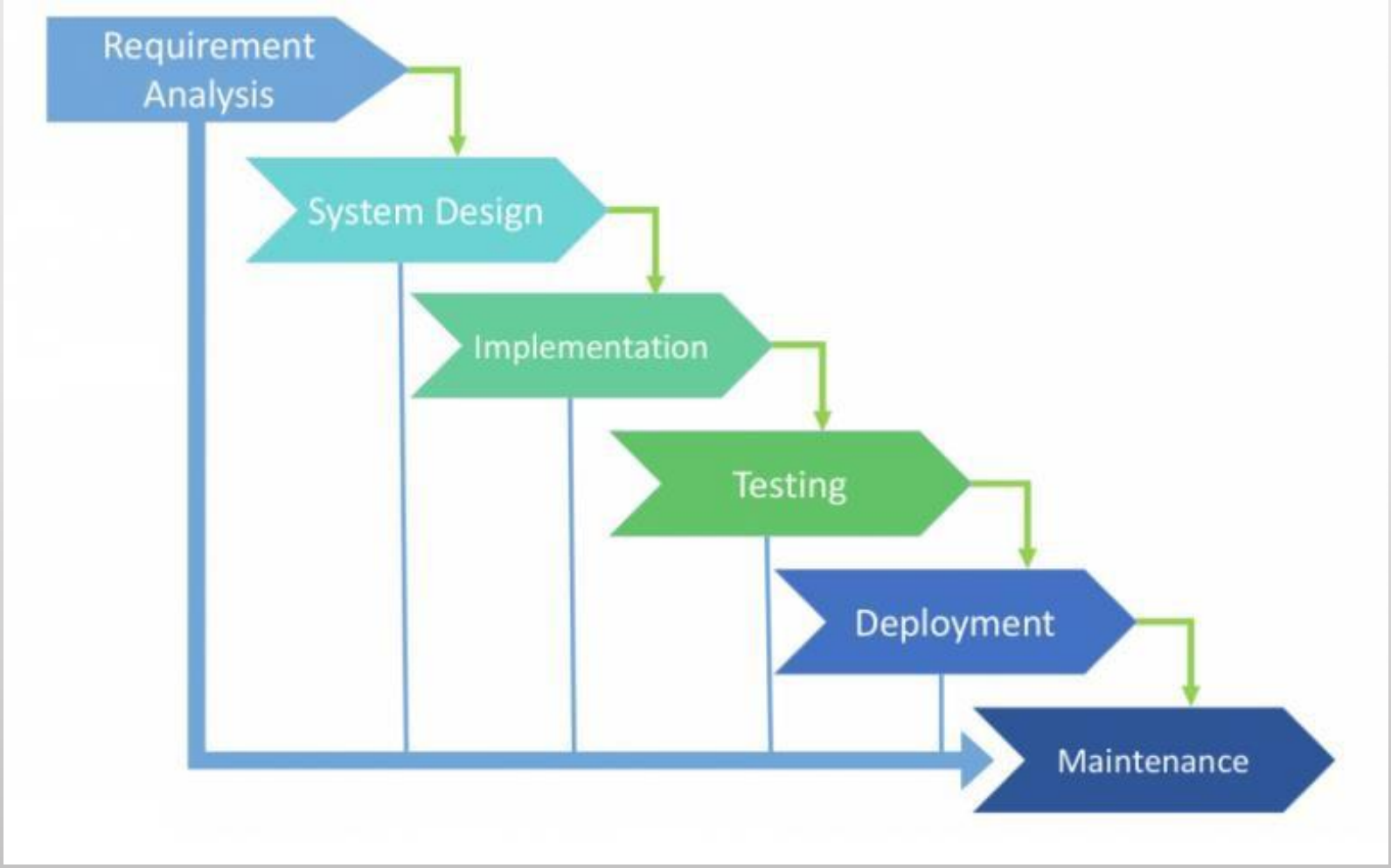
Assignments

Sequence [**Todays Agenda**]

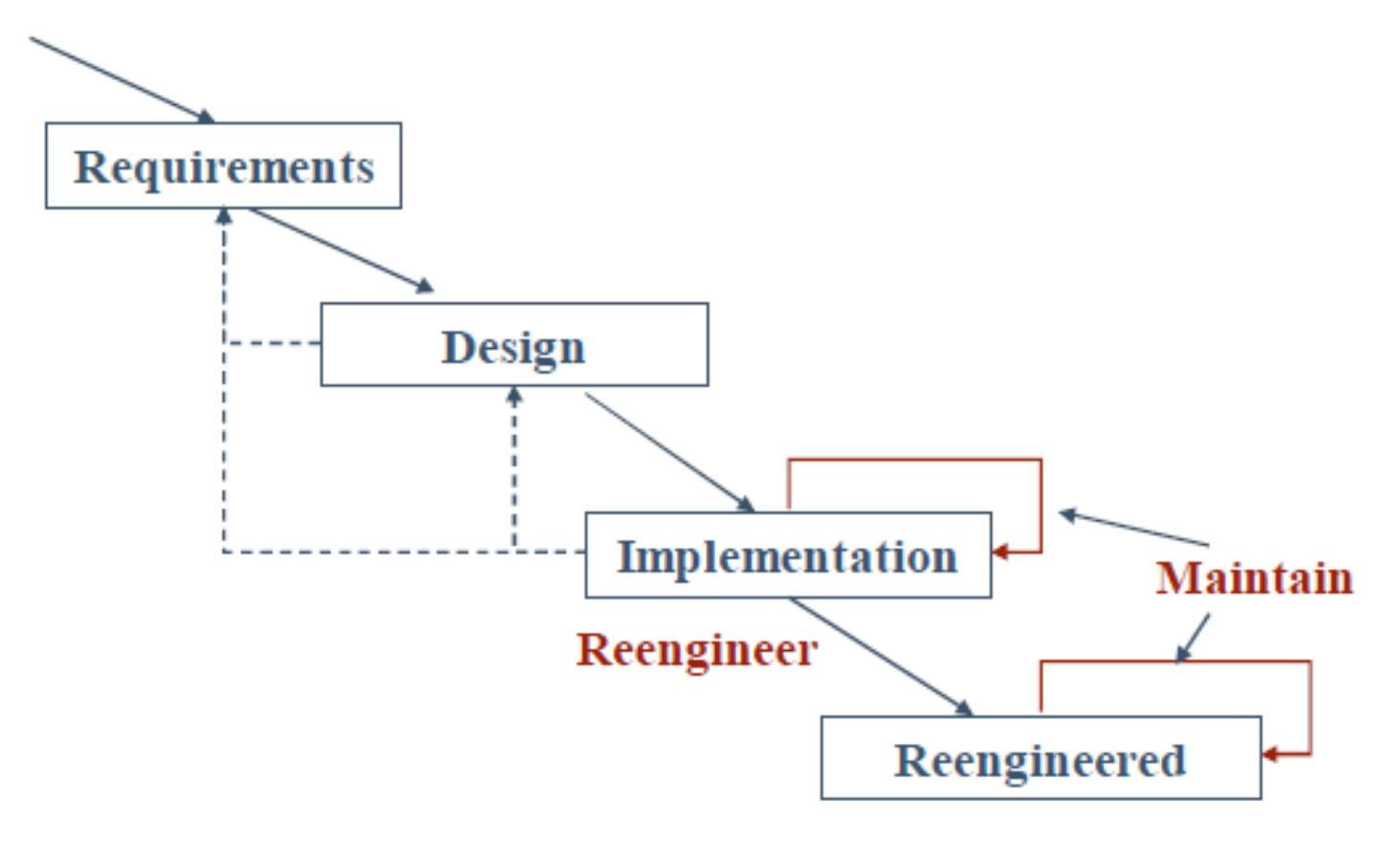
Content of Lecture

- Introduction, Context and Strategies
- State-of-the-art definitions
- Reengineering difficulties and challenges
- Reengineering Techniques
- Why Reengineering Projects Fail

Traditional SDLC



Context: Software Life Cycle



Software Reengineering – Introduction - 1

- Software Reengineering is the process of **improving the quality** of the software product.
- Software reengineering encompasses all activities related to **restructuring**, **re-analyzing**, **re-documentation** of already existing software program into a new one.
- Software reengineering is used to update the existing software in the new form product so that the software product will provide **high performance**, and **improve the functionality** of the system.
- Software reengineering is the combination of **reverse engineering**, **re-documentation**, **restructuring**, **translation**, and **forward engineering**.

Software Reengineering – Introduction - 2

- The reason we need **timely changes** in our software system is the fact that there are so many **new technologies emerging** everyday.
- In other words, software reengineering is an approach of **updating** software **without changing its working**.
- This process may be done by adding some **new features** and **functionalities** in the software product to make the software more **efficient** and **reliable**.

Technical Definition:

*Software reengineering is the **examination** and **alteration** of a system to **reconstitute** it in a new form.*

Software Reengineering – Introduction - 3

- In the reengineering process, the system is **documented** and **organized** again.
- It may be **transformed** into a **present programming language** and executed on **current hardware technology**.
- Therefore, software reengineering helps in:
  - *Translating source code into a new language*
  - *Rearrange the old source code*
  - *Transferring them to a new platform (client-server)*
  - *Acquiring information about them*
  - *And re-documenting systems which are badly documented*

Reengineering – Definitions - 1

Several definitions of reengineering:

- Preparation or improvement to software, usually for increased maintainability, reusability or evolvability.

R. S. Arnold, *A Roadmap Guide to Software Reengineering Technology*, Software Reengineering, 1994.

- The examination and alteration of a subject system to reconstitute it in a new form and the subsequent implementation of the new form

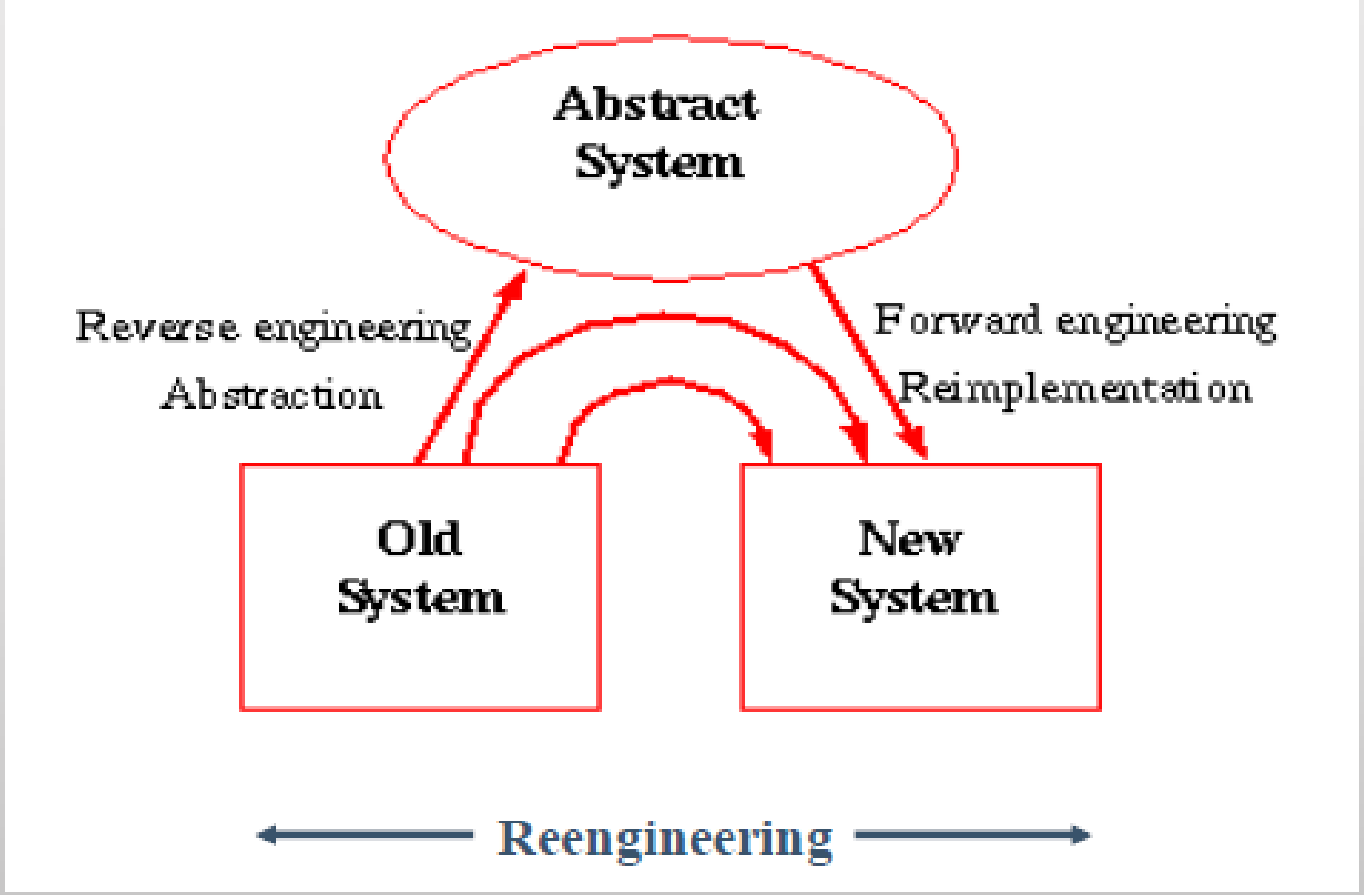
E. Chikofsky and J. Cross, Reverse Engineering and Design Recovery: A Taxonomy, IEEE Software 7(1) Jan 1990: 13-17.

Reengineering – Definitions - 2

- Reengineering is the systematic transformation of an existing system into a new form to realize quality improvements in operation, system capability, functionality, performance, or evolvability at a lower cost, schedule, or risk to the customer.

S. Tilley and D. Smith, Perspectives on Legacy System Reengineering, SEI White Paper, 1995

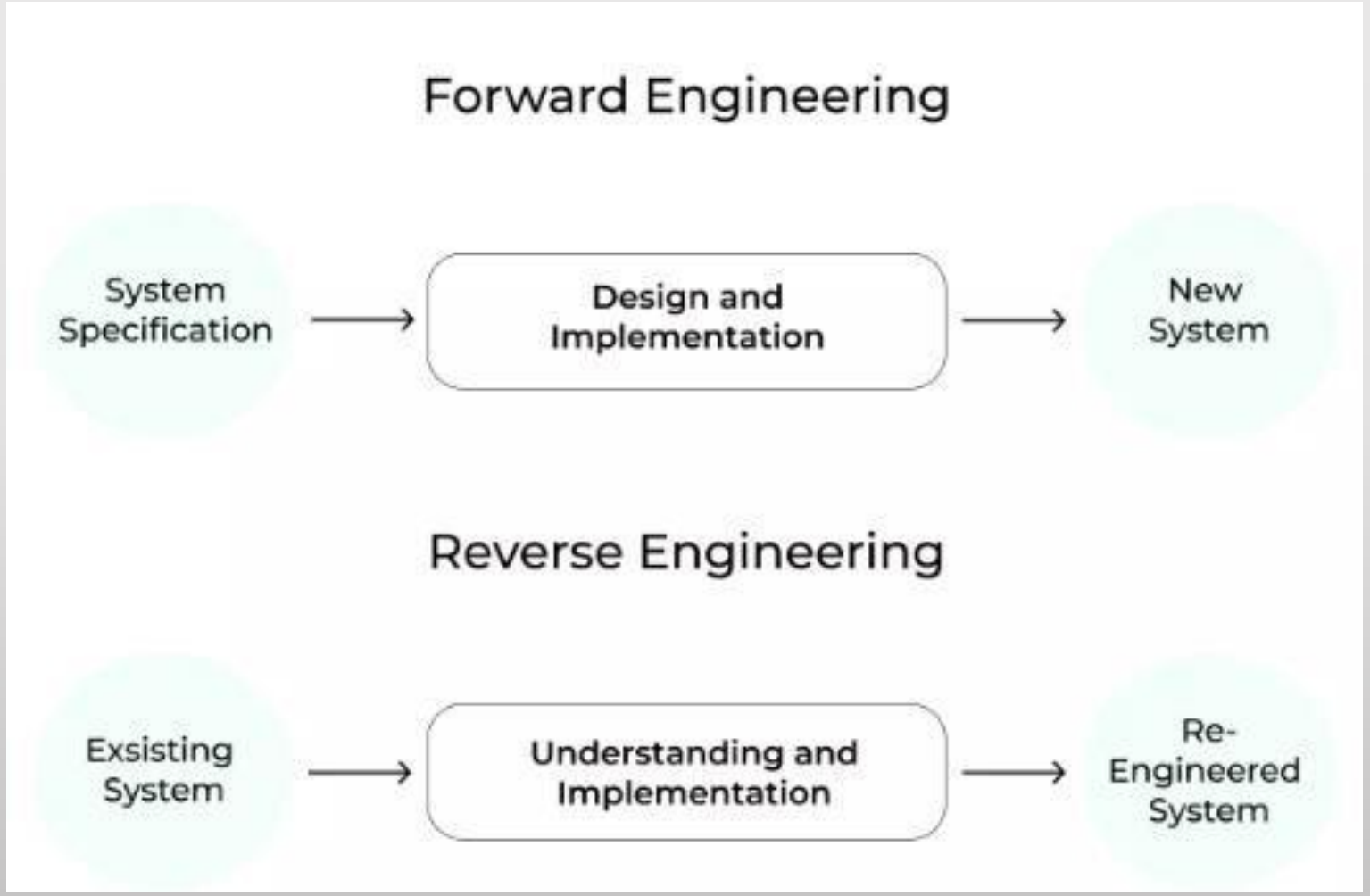
What is Reengineering?



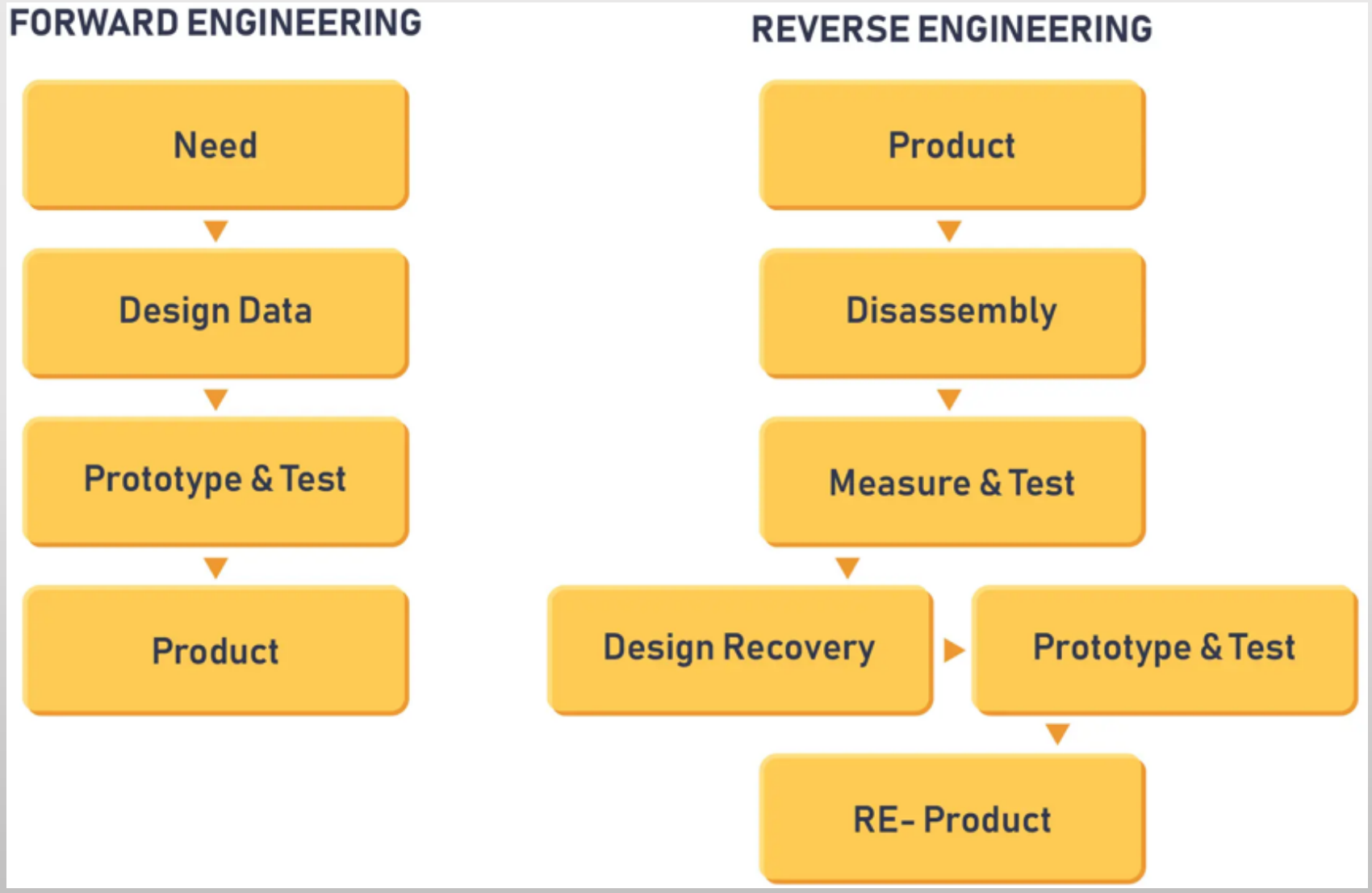
Forward VS Reverse Engineering

- A new software is created by going downward from the top, highest level of abstraction to the bottom, lowest level. This downward movement is known 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:
  - Analyze the software to determine its components and the relationships among the components.
  - 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.

Forward VS Reverse Engineering



Forward VS Reverse Engineering



Reengineering – Related Topics - 1

Restructuring:

- Restructuring is the transformation from one representation form to another at the same relative abstraction level, while preserving the subject system's external behavior

Refactoring:

- Refactoring is the process of changing a software system in such a way that it does not alter the external behavior of the code yet improves its internal structure.

Reengineering – Related Topics - 2

Business Process Reengineering (BPR):

- concerned with the conversion of business process and not of computer processes

Data Reengineering:

- concerned with conversion of data format and not its content

Why Reengineer? - 1

Why does an organization decide to reengineer one or more of their systems:

- Software system is an integral part of the organization.
- System must be regularly maintained, and maintenance is becoming a large cost factor.
- Less risk and cost than redevelopment.

Why Reengineer? - 2

- Emergence of new technologies
- Increasing competition against companies
- Quality of the product is gaining importance.
- Needs of people changes.
- It is sometime a cost effective option for software evolution.
- Applicable when some (but not all) subsystems of a larger system need frequent maintenance.
- It involve putting in the effort to make it easier to maintain.
- Software re-engineering allows to reusability of the software products.

Why Reengineer? - 3

- When we need to update the software to keep it to the current market, without impacting its functionality, it is called software re-engineering.
- It is a thorough process where the design is changed and programs are re-written.
- Legacy software cannot keep tune with the latest technology in the market. As the hardware become obsolete, updating of software becomes a overhead.
- Even if software grows old with time, its functionality does not.
- *For example: Linux was developed in assembly language. When C language came into existence, Linux was re-engineered in C, because working with assembly language was difficult.*

Why Reengineer? - 4

- Other than this, sometimes programmers noticed that few parts of the software needs more maintenance than others and need re-engineering.

Software Reengineering – Cost

- Costs are affected by the following factors:
  - The quality of the software to be re-engineered.
  - The tool support available for the re-engineering.
  - The extent of the required data conversion.
  - The availability of expert staff for re-engineering.

Reengineering difficulties and challenges

- Reasons why legacy systems have not been designed to accommodate change:
- Short life expectancy – not anticipated to last decades when first developed.
- Failure of process models and software engineering culture to treat evolution as a first class activity – future requirements ignored.
- Satisfying constraints that existed at the time of development – hardware (memory and processing power).

Reengineering Strategies

Reengineering Strategies for legacy applications:

- Ignore – discard them
- Cold turkey – rewrite them from scratch
- Integrate – consolidate them into the current and future applications by access in place
- Data Warehouse – build a “shadow” system to house the frequently accessed data
- Gradual Migration – rearchitect and transition gradually

Factors Influencing Strategy

The following factors will influence what strategy is to be chosen:

- Business value of the legacy systems – support current as well as future business processes.
- Flexibility and growth requirements – if legacy system does not need extensive changes for flexibility and growth then minimal effort for integration may be appropriate.
- Technical status of the legacy system – represents the quality of the system in terms of modularity, error rates, flexibility and utilization of current technologies.

Categories of Legacy Applications

- Completely decomposable – all components are well structured and separable. Friendly to migration efforts
- Data decomposable – data services well structured and well-defined interfaces. Semi-friendly to migration efforts
- Program decomposable – program modules are well structured and have well-defined interfaces. Semi-friendly to migration efforts

Strategies – Ignore - 1

- When can the existing system be discarded?

- The business has changed and new methods of doing business have been developed, existing methods are being phased out.

- It is too costly to try to reengineer the current systems.

- Current systems are monolithic and not reusable.

- New COTS products can be bought with the same or similar functionality for less cost that reengineering

Strategies – Ignore - 2

- In most situations it is not practical to ignore legacy systems especially the data.
- Some organizations are ignoring legacy systems by outsourcing them to software houses that specialize in operating and maintaining legacy applications.

Strategies – Cold Turkey - 1

When can the existing system be ignored?

- The business has changed and new methods of doing business have been developed – need to build systems to implement these new methods.
- It is too costly to try and understand what you currently have.
- It is too costly to try to reengineer the current systems.
- Time and resources are available to start from scratch.

Strategies – Cold Turkey - 2

In practice this approach may not be viable because:

- Management may not allow such large investments in development just to have a more flexible system – must have a better system produced.
- Business does not stand still while new systems are being developed.
- Specifications for legacy systems rarely exist and undocumented dependencies frequently exist.
- The rewritten system themselves become legacy before completion as development takes a long time.

Strategy – Integrate - 1

When can the existing system be integrated?

- A lot of the business knowledge and rules are built into the current systems and these need to be preserved – high business value.
- Costs will be reduced by reengineering the current systems rather than developing from scratch.
- Time and resources are not available to start from scratch.

Strategy – Integrate - 2

In practice integration is carried out when:

- Access to needed data is urgent and cannot be postponed until completion of migration.
- Needed data can be accessed by client applications by employing COTS technologies.

Strategy – Data Warehouse - 1

- When can the data warehouse option be used?:
- The data is the most important part of the existing systems.
- There is a need for new ways to access and manipulate the data (web-based, distributed).
- There isn't a need to reimplement existing system functionality and time and resources are not available to do so.

Strategy – Data Warehouse - 2

- Organization's data is in two formats:
- Operational data which is used for day-to-day transaction processing.
- Analysis (decision support) data which are used for business analysis and report generation.

This data is extracted from the operational data periodically and downloaded, usually to a separate system, for report generation and analysis.

Strategies - Gradual Migration - 1

When can the gradual migration option be used?:

- Integration is not economical over a long period of time or those legacy systems are to be phased out.
- There is an immediate need to update some of the more important components.
- Time and resources are not available to do the reengineering all at once.
- It is unclear how the reengineering effort will proceed – need for prototyping of different reengineering methods.

Strategies - Gradual Migration - 2

- The legacy and target systems may coexist during the migration stage.
- Challenge will be to design gateways to isolate the migration steps so that end users do not know if the data is coming from old or new systems.
- The development of gateways to facilitate migration is usually an expensive undertaking.

Reengineering Issues - 1

- Ease of change
  - how difficult will it be to change the legacy system
- Comprehensibility
  - how difficult is it to obtain an understanding of the systems that will need to be reengineered
- Size
  - what is the size of the systems that have to be reengineered
- Decomposability
  - how easy or difficult is it to decompose the software system

Reengineering Issues - 2

- Degree of coupling
  - how coupled are the existing components
- Degree of cohesion
  - how cohesive are the components
- Granularity
  - can the system be reused in large chunks or does it need to be decomposed
- Complexity
  - what is the level of complexity of the software
- Degree of documentation
  - what documentation is available

Reengineering Issues - 3

- Degree of abstraction
  - what abstractions are available
- Language type & choice
  - what language(s) is the existing system written in and what will be used in the new system
- Referential transparency
  - do the components always produce the same output

Reengineering – Risks - 1

- Some of the risk areas in reengineering :-
- Process
  - is there a defined process to follow to successfully reengineer the systems.
- Personnel
  - are the necessary resources available.
- Skills/Education
  - do the personnel have the right skills or can they be trained.
- System/Application
  - difficulties with the legacy system.

Reengineering – Risks - 2

- Technology

  what technology is used in the target system
- Tools

  are there tools to assist in the reengineering work
- Strategy

  has the right strategy been chosen

Reengineering Projects – Why they fail?

- Reengineering projects fail for many reasons:
- Most reasons are not specific to reengineering projects but also occur in new development projects.
- Usually not just one reason why a project fails but maybe a combination of reasons.
  - **Bergey, John; Smith, Dennis; Tilley, Scott; Weiderman, Nelson; Woods, Steven. *Why Reengineering Projects Fail* (CMU/SEI-99-TR-010).**

Reengineering Projects Fail - 1

Reason #1:

- The organization inadvertently adopts a flawed or incomplete reengineering strategy.
- This may be caused by poor assumptions or lack of attention to detail. Maybe the wrong problem is being addressed. Possibly not all of the components and steps are considered. If a system is ignored or discarded a lot of corporate knowledge may be lost/abandoned.

Reengineering Projects Fail - 2

Reason #2:

- The organization makes inappropriate use of outside consultants and outside contractors.
- Outsiders may bring domain understanding, technical skills, and extra resources. However they may not know the business as well as the insiders. Their role needs to be clearly defined and monitored. Outsiders may have conflicting interests (maximizing cost rather than minimizing). Outsiders take control of the work rather than it being controlled by insiders (results in lack of insight by insiders)

Reengineering Projects Fail - 3

Reason #3:

- The work force is tied to old technologies with inadequate training programs.
- The lack of training can cause project failure. New programming paradigms will be adopted. No possible to continue to do business as usual while at the same time bringing the same workforce up to speed on new technologies. Must be a conscientious and persistent effort to upgrade skills of the existing workforce or there must be a replacement of existing workforce or replacement or some combination.

Reengineering Projects Fail - 4

- Reason #4:
- The organization does not have its legacy system under control.
- Before a system can be managed effectively, a system baseline under configuration management should be in place to aid in disciplined evolution. Legacy system needs to be well documented, with an understanding of the priority of change requests and their impact on the system.

Reengineering Projects Fail - 5

- Reason #5:
- There is too little elicitation and validation of requirements for the reengineering effort.
- There may be flaws in the requirements elicitation and validation processes. Requirements include functional, non- functional, user and customer requirements.

Reengineering Projects Fail - 6

Reason #6:

- Software architecture is not a primary reengineering consideration.
- Failure can occur when a methodical evaluation of the software architecture of the legacy and target systems is not the driving factor in the development of the reengineering technical approach. Evaluation of the legacy architecture may decide what strategy is undertaken.

Reengineering Projects Fail - 7

Reason #7:

- There is no notion of a separate and distinct reengineering process.
- The means by which a legacy system evolves can have a large influence on the success of failure. The existence of a documented life-cycle process and corresponding work products are often wrongly viewed as being evidence of a sound reengineering process. There needs to be a set of tasks and guidance to perform them and an understanding of how it all fits together.

Reengineering Projects Fail - 8

Reason #8:

- There is inadequate planning or inadequate resolve to follow the plans.
- Projects often get out of kilter by focusing on the low-level "software problems" and neglecting the intermediate-level tactical management planning and systems engineering planning aspects of the job.
- Sometimes there may be an absence of a documented project plan that has the buy-in of the key stakeholders (line management, project team, domain and system experts and software engineers).

Reengineering Projects Fail - 9

Reason #9:

- Management lacks long-term commitment.
- Management support means careful monitoring and putting things back on the track when they stray off track.
- If management gets distracted with other projects during the course of a major reengineering effort, it will not know when things go wrong.

Reengineering Projects Fail - 10

Reason #10:

- Management predetermines technical decisions.
- Mandates or edicts issued by the upper management that predetermine the technical approach or schedule, cost, and performance considerations without sufficient project team input or concurrence are frequently seen to cause reengineering project failure.

Commercial Off-the-Shelf Software (COTS)

- Commercial Off-the-Shelf Software (COTS) is software that is commercially produced
- COTS are ready to use without any form of modification by the user, and accessible to everyone.
- Example: SAP, Oracle etc.

Monolithic Architecture in Software Reengineering

- Monolithic architecture is the traditional unified model for the design of software program.
- Monolithic, in this context, means "composed all in one piece."
- The various functions are tightly coupled rather than loosely coupled, like in modular software programs, and several such functions are part of a single application.
- For all these reasons, monolithic software is usually very complex.

Cold Turkey

- Involves rewriting a legacy IS from scratch to produce the target IS using modern software techniques and hardware of the target environment.
- The phrase "going cold turkey" is commonly used to describe the suddenly stopping of a habit, particularly in the context of quitting addictive substances like drugs or alcohol.
- The origins of the expression are somewhat believed to have emerged in the early 20th century.

Chicken Little Approach in Software Reengineering

- An approach that allows the coexistence of the legacy and target databases during the data reengineering phase through the use of a gateway
- Involves migrating the legacy IS in place, by small incremental steps until the desired long term objective is reached.

Data warehouse:

- A data warehouse is a system that aggregates (collection, sum up, total) data from multiple sources into a single, central and consistent data store.
- Data warehouses help prepare data for data analytics, business intelligence (BI), data mining, machine learning (ML) and artificial intelligence (AI) initiatives.

Data Manipulation

- Data manipulation tools can modify data to make it easier to read or organize.
- These tools enable users to identify patterns in data that may otherwise not be obvious.
- Techniques:
  - Filtering
  - Aggregation
  - Data Cleaning
  - Data Analysis
  - Handling missing data
  - Sorting
  - Data Transformation

Coupling

- Coupling refers to the degree of interdependence between software modules.
- High coupling means that modules are closely connected and changes in one module may affect other modules.
- Low coupling means that modules are independent, and changes in one module have little impact on other modules.

Cohesion

- Cohesion in software engineering is a fundamental concept that refers to the degree of interdependence among the different components or modules within a software system.
- It measures how closely related the functionalities or responsibilities of these components are to each other.
- It refers to the degree to which elements within a module work together to fulfill a single purpose.
- High cohesion means that elements are closely related and focused on a single purpose, while low cohesion means that elements are loosely related and serve multiple purposes.

- Both coupling and cohesion are important factors in determining the maintainability, scalability, and reliability of a system.
- In legacy systems high coupling and low cohesion can make a system difficult to change and test, while low coupling and high cohesion make a system easier to maintain and improve.

Referential Transparency

- Referential transparency refers to computable expression.
- A piece of code is referentially transparent if we can safely replace that piece of code with the value it computes and vice-versa, anywhere where that piece is used, without changing the meaning or result of our program

- The following strategies can be used when reengineering a software system:
  - Upgrading source code to the newest language version.
  - Step by step modification of old components to new ones
  - Creating a completely new system
  - Creating a new system considering new and/or updated business processes.


Thank You!