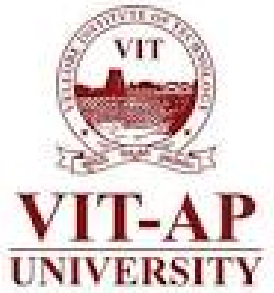
CSE1005: Software Engineering

Module No. 6: Software Quality and Maintenance



Dr. K Srinivasa Reddy
SCOPE, VIT-AP University

Module No. 6: Software Quality and Maintenance

Software Quality: Software Quality Factors, Verification & Validation, Software Quality Assurance, The Capability Maturity Model

Software Maintenance: Software maintenance, Maintenance Process Models, Maintenance Cost, Reengineering, Reengineering activities, Software Reuse.

Text Book:

1. Roger Pressman, “Software Engineering: A Practitioner’s Approach”, McGraw-Hill, 7th Edition, 2016.

Course Outcome: Understand the factors of software quality and the activities of maintenance of software (CO6)

Software Quality - Garvin's Quality Dimensions - Example

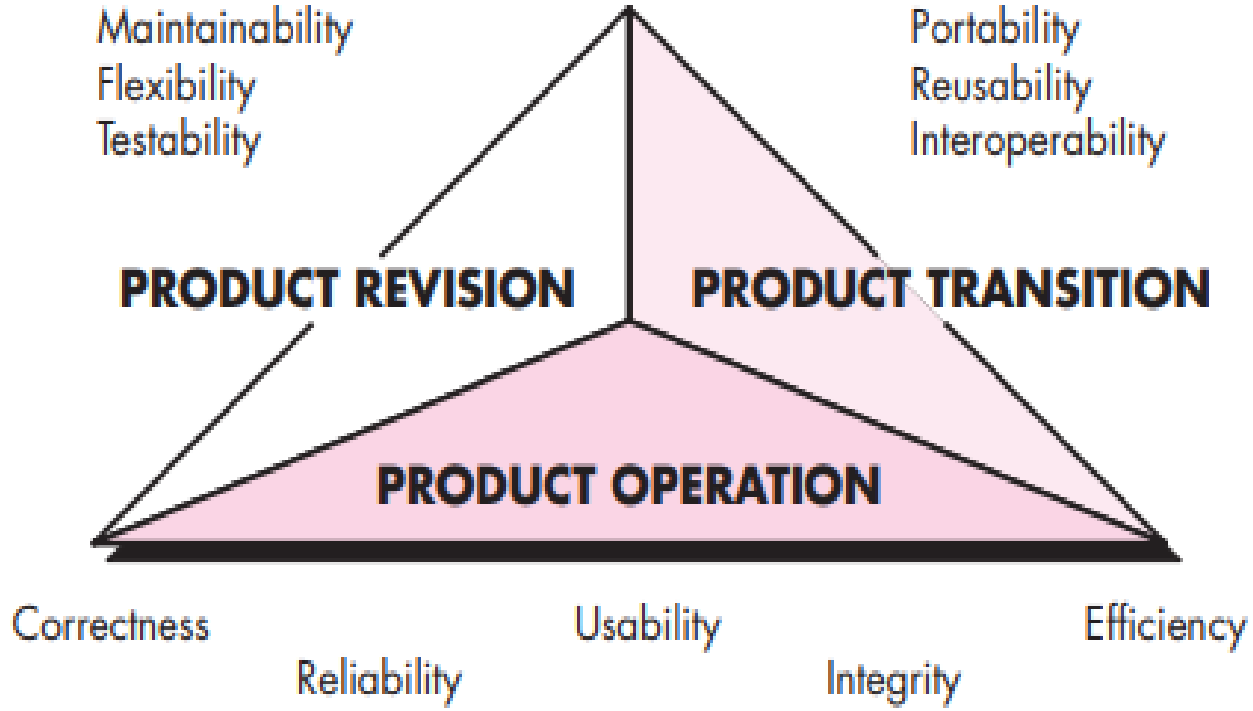
1. **Performance -** Millage per liter, acceleration, handling, cruising speed etc.
2. **Features -** Built-in GPS, seat warmer, smartphone integration etc.
3. **Reliability –** For a New car, do not expect the vehicle to break down frequently.
4. **Conformance -** Degree to which a product conforms to its specification.
5. **Durability -** How long will your car last?
    1. Reliability is the probability of failure over a specified period. Durability is the number of cycles or the time a component will function properly as a part of the product life.
6. **Serviceability -** How much time and effort it takes to get a faulty product repaired and returned to regular use.
7. **Aesthetics -** The weight, color, size, texture, packaging design etc.
8. **Perceived quality -** When was the last time you had a positive experience with a car show room or service center? It's the combined effects of factors such as brand name, price, salesperson, marketing strategy etc.

Software Quality

- An effective software process applied in a manner that creates a useful product that provides measurable value for those who produce it and those who use it.
- **Garvin's Quality Dimensions**
  1. **Performance -** Product's essential function
  2. **Features -** Extra functionality to their essential functions
  3. **Reliability -** Ability of a product or service to perform as expected over time.
  4. **Conformance -** Degree to which a product conforms to its specification.
  5. **Durability -** The measurement of product life, i.e., the amount of use the customer could get from the product before it deteriorates.
  6. **Serviceability -** The ease at which a user can repair a faulty product or get it fixed.
  7. **Aesthetics -** The appearance of a product or service.
  8. **Perceived quality -** Overall opinion of the customers towards the product.

Software Quality - Software Quality Factors - McCall's Quality Factors

- **McCall, Richards, and Walters** propose a useful categorization of factors that affect software quality
- These factors **focus on three important aspects of a software**:
  - Operational characteristics
  - Ability to undergo change - Revision
  - Adaptability to new environments - Transition



Software Quality - Software Quality Factors - McCall's Quality Factors

- **Operational characteristics**
  - **Correctness:** Extent to which a program satisfies specification and fulfills the customer's objectives.
  - **Reliability:** Extent to which a program can be expected to perform its intended function with required precision.
  - **Usability:** Effort required to learn, operate, prepare input for, and interpret output of a program.
  - **Efficiency:** The amount of computing resources and code required by a program
  - **Integrity:** Extent to which access to software or data by unauthorized persons can be controlled.
- **Revision characteristics**
  - **Maintainability:** Effort required to locate and fix an error in a program.
  - **Flexibility:** Effort required to modify an operational program.
  - **Testability:** Effort required to test a program.

Software Quality - Software Quality Factors - McCall's Quality Factors

- **Transition characteristics**
  - **Portability:** Effort required to transfer the program from one hardware and/or software system environment to another.
  - **Reusability:** Extent to which a program [or parts] can be reused in other applications
  - **Interoperability:** Effort required to couple one system to another

Software Quality Assurance (SQA) Elements

- Also called as **Quality Management**
- A process that **assures** that all software engineering processes, methods, activities, and work items are monitored and comply with the defined standards.
- These defined standards could be one or a combination of any like ISO 9000, CMMI model, ISO15504, etc.
- Takes place **before, during, and after** the **software development life cycle.**
- **Its prime goal is to ensure quality.**
- Software quality assurance (SQA) encompasses
  - An SQA process
  - Specific quality assurance and quality control tasks (including technical reviews and a multitiered testing strategy)
  - Effective software engineering practice (methods and tools)
  - Control of all software work products and the changes made to them
  - A procedure to ensure compliance with software development standards (when applicable)
  - Measurement and reporting mechanisms

Software Quality Assurance (SQA)

SQA Tasks

- **Prepares** an SQA plan for a project
- **Participates** in the development of the project's software process description
- **Reviews** software engineering activities to verify compliance with the defined software process
- **Audits** designated software work products to verify compliance with those defined as part of the software process
- **Ensures** that deviations in software work and work products are documented and handled according to a documented procedure
- **Records** any noncompliance and reports to senior management

Software Quality Assurance

VIT-AP UNIVERSITY

The SQA task are performed to achieve a set of pragmatic goals:

- **Requirements quality –** Correctness, completeness, and consistency of the requirements model
- **Design quality –** Model should be assessed by the software team
- **Code quality –** Source code and related work products (e.g., other descriptive information) must conform to local coding standards and exhibit characteristics that will facilitate maintainability
- **Quality control effectiveness –** Team should apply limited resources in a way that has the highest likelihood of achieving a high-quality result

The Capability Maturity Model

VIT-AP UNIVERSITY

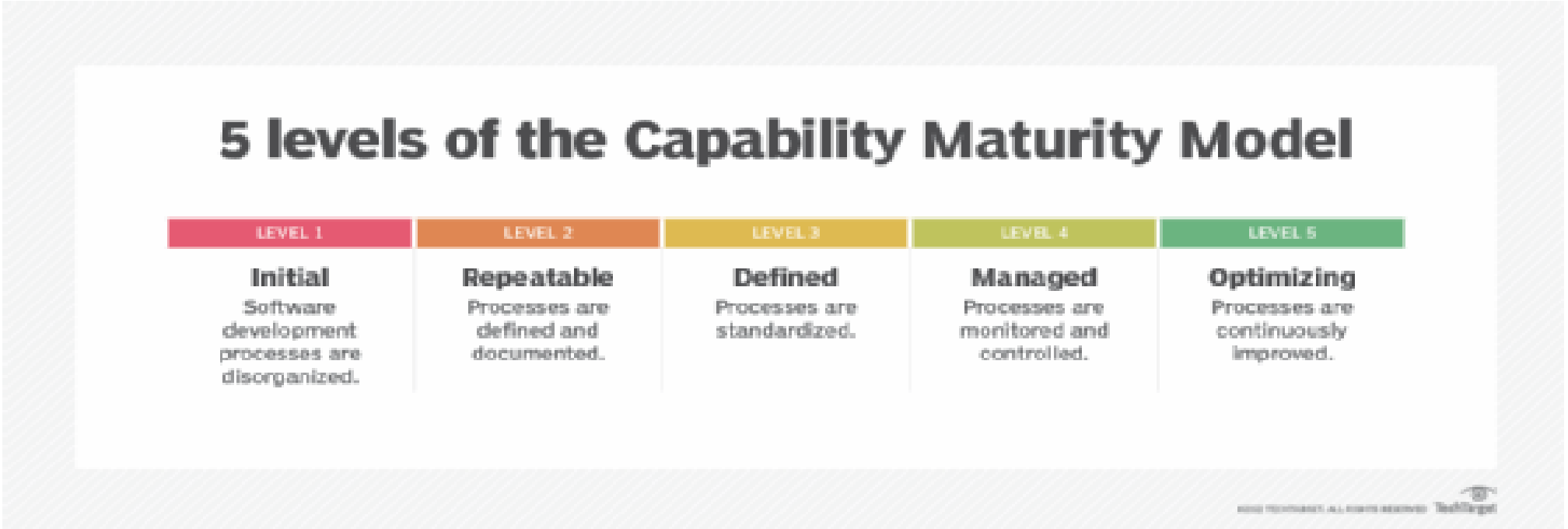
- Developed by the **Software Engineering Institute (SEI)** of the Carnegie Mellon University
- A framework that **describes the key elements** of an effective software process.
- Describes an **evolutionary improvement path from** an **ad hoc**, **immature process to a mature, disciplined process**.
- **Provides guidance** on
  - **How to gain control of processes for developing and maintaining software** and
  - **How to evolve toward a culture of software engineering and management excellence.**

The Capability Maturity Model : Components of CMM

VIT-AP UNIVERSITY

Levels of software process maturity : 05

- Maturity level indicates level of process capability:
  - Initial
  - Repeatable
  - Defined
  - Managed
  - Optimizing



The Capability Maturity Model

- **Level 1 Initial:**
  - Defines a newbie organization that does not have a process yet **nor** does it have a project-tracking system that enables developers to predict costs or finish dates with any accuracy.
  - Few processes are defined, and success **depends more on individual heroic efforts than on following a process** and **using a synergistic team effort**.
- **Level 2 Repeat:**
  - Defines an organization that **repeats some of the processes**
  - Basic project management processes are established to track cost, schedule, and functionality.
  - Planning and managing new products is based on experience with similar projects**.**
  - But there is no consistency or coordination among different groups.

The Capability Maturity Model

- **Level 3 Defined:**
  - Defines an organization having a set of the standard processes for management and engineering are documented, standardized, and integrated into a standard software process for the organization.
  - All projects use an approved, tailored version of the organization's standard software process for developing software
- **Level 4, Managed:**
  - Defines an organization having control over these processes.
  - Detailed software process and product quality metrics establish the quantitative evaluation foundation.
  - Meaningful variations in process performance can be distinguished from random noise, and trends in process and product qualities can be predicted
- **Level 5, Optimized:**
  - The organization has quantitative feedback systems in place to identify process weaknesses and strengthen them pro-actively.
  - Project teams analyze defects to determine their causes; software processes are evaluated and updated to prevent known types of defects from recurring.

The Capability Maturity Model

VIT-AP UNIVERSITY

- The Capability Maturity Model represents the staged path for an organization's performance growth and process improvement efforts based on a predefined set of practice areas.
- With each passing maturity level, the predefined set of practice areas provide organizations with a clear path for improvement, even at the highest maturity level.
- Each maturity level builds on the previous, adding layers of new functionality that improve processes dramatically.

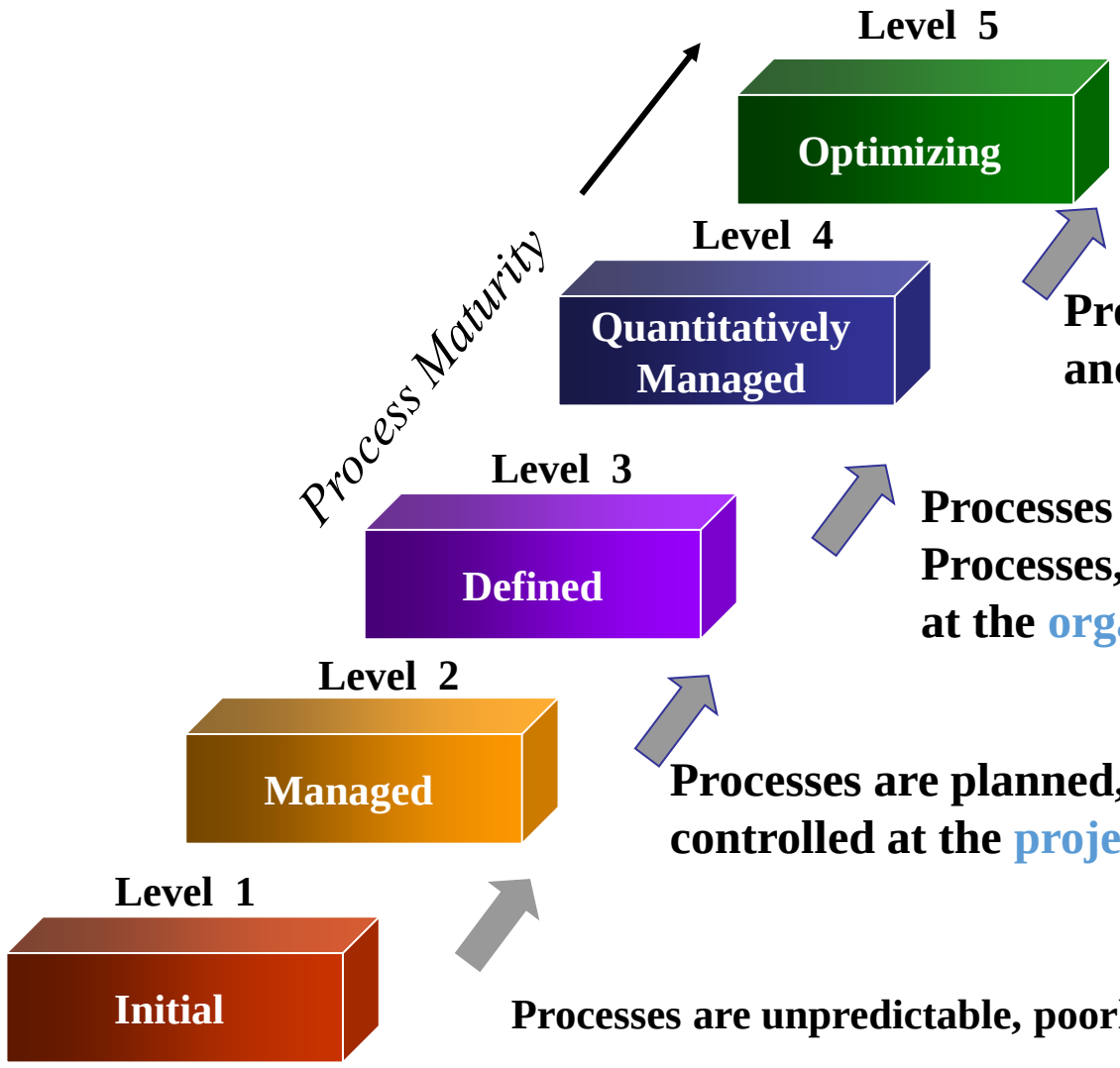
Capability Maturity Model Integration (CMMI)

VIT-AP UNIVERSITY

- CMMI (Capability Maturity Model Integration) is a proven industry framework to improve product quality and development efficiency for both hardware and software

- The main difference between CMM (Capability Maturity Model) and CMMI (Capability Maturity Model Integration) is that the former focuses on evaluating whether an organization completes specific tasks related to the process or not, while the latter focuses on building an architecture for the whole development process.

Capability Maturity Model Integration (CMMI)



Process performance continually improved through incremental and innovative technological improvements.

Processes are controlled using statistical and other quantitative techniques.

Processes are well characterized and understood. Processes, standards, procedures, tools, etc. are defined at the organizational (Organization X) level. Proactive.

Processes are planned, documented, performed, monitored, and controlled at the project level. Often reactive.

Processes are unpredictable, poorly controlled, reactive.

Difference Between CMM and CMMI

Parameters of Comparison	CMM	CMMI
Definition	Capability Maturity Model.	Capability Maturity Model Integration.
Meaning	It is a behavior model developed to measure an organization's software development process.	It is a successor of the CMM model and is more effective and task-oriented.
Developed in	The year 1990	The year 2006
Purpose	To evaluate the process maturity levels in software.	To combine many software models into one and to overcome the drawbacks of CMM.
Stages	This model has five stages: Initial, Repeat, Defined, Managed, Optimized.	This model has five stages, too: Initial, Managed, Defined, Qu antitively managed, Optimized.
Efficiency	Less effective one	More effective one

Software Maintenance:

VIT-AP UNIVERSITY

- Software maintenance is a part of the Software Development Life Cycle.
- **Primary goal** is to modify and update software application after delivery to correct errors and to improve performance.
- Software is a model of the real world.
  - When the real world **changes**, the software **require alteration wherever possible.**
- **Software Maintenance** is an inclusive activity that includes
  - Error corrections,
  - Enhancement of capabilities,
  - Deletion of obsolete capabilities, and
  - Optimization.

Software Maintenance:

- Software Maintenance is needed for:-
  - Correct errors
  - Change in user requirement with time
  - Changing hardware/software requirements
  - To improve system efficiency
  - To optimize the code to run faster
  - To modify the components
  - To reduce any unwanted side effects.
  - Thus the maintenance is required to ensure that the system continues to satisfy user requirements.

Software Maintenance:

- CATEGORIES OF SOFTWARE MAINTENANCE • There are four types of software maintenance:

1. **Corrective maintenance :** This refer to modifications initiated by defects in the software.
2. **Adaptive maintenance:** It includes modifying the software to match changes in the ever changing environment.
3. **Perfective maintenance:** It means improving processing efficiency or performance, or restructuring the software to improve changeability. This may include enhancement of existing system functionality, improvement in computational efficiency etc.
4. **Preventive maintenance:** It is the process by which we prevent our system from being obsolete. It involves the concept of reengineering & reverse engineering in which an old system with an old technology is re-engineered using new technology. This maintenance prevents the system from dying out.

Software Re-engineering

Example: The rebuilding of a house.

Consider the following situation. You've purchased a house in another state. You've never actually seen the property, but you acquired it at an amazingly low price, with the warning that it might have to be completely rebuilt. How would you proceed?

Example: The rebuilding of a house

- Before you can start rebuilding, it would seem reasonable to inspect the house.
- Before you tear down and rebuild the entire house, be sure that the structure is weak.
  - If the house is structurally sound, it may be possible to "remodel" without rebuilding (at much lower cost and in much less time).
- Before you start rebuilding be sure you understand how the original was built.
  - Understand the wiring, the plumbing, and the structural internals.
  - Even if you trash them all, the insight you'll gain will serve you well when you start construction.
- If you begin to rebuild, use only the most modern, long-lasting materials.
- If you decide to rebuild, be disciplined about it. Use practices that will result in high quality-today and in the future.

Software Re-engineering

- Although these principles focus on the rebuilding of a house, they apply equally well to the reengineering of computer-based systems and applications. To implement these principles, you can use a software reengineering process model.

Legacy System:

- A system is called a legacy system when it has one or all of the following properties
  - Evolved over 10 - 30 years
  - Actively used in a production environment
  - Considered irreplaceable because reimplementation is too expensive or impossible
  - Very high maintenance cost
  - Designed without modern software design methodologies or design has been "lost".
- Unmaintainable software is not a new problem. In fact, the broadening emphasis on software reengineering has been spawned by software maintenance problems that have been building for more than four decades.

Software Re-engineering

VIT-AP UNIVERSITY

Introduction to Re-engineering

- Re-engineering means to re-implement systems to make more maintainable.
- *Re-engineering is the reorganizing and modifying existing software systems to make them more maintainable*
- In this, the functionality & architecture of the system remains the same but it includes redocumenting, organizing, modifying & updating the system.
- When re-engineering principles are applied to business process then it is called **Business Process Reengineering (BPR).**

Steps in Re-engineering

- The steps involved in re-engineering are-
  - Goal setting
  - Critical analysis of existing scenario such as process, task, design, methods etc.
  - Identifying the problems & solving them by new innovative thinking. In other words, it uses new technology to gain a significant breakthrough improvement.

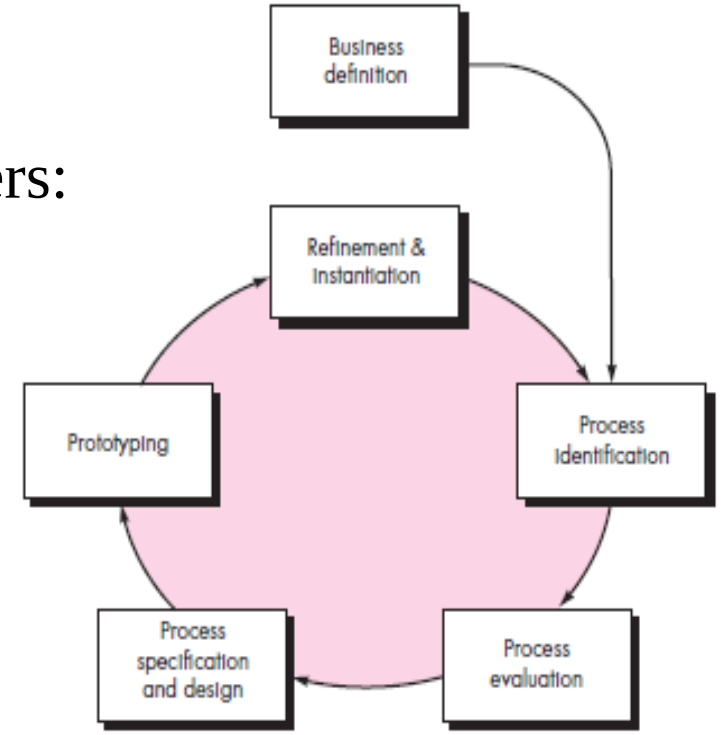
Software Re-engineering

Business Process Reengineering (BPR) Model:

- Business process reengineering is iterative.
- Business goals and the processes that achieve them must be adapted to a changing business environment.
- For this reason, there is no start and end to BPR - it is an evolutionary process.
- The model defines six activities:

1. Business definition:

- Business goals are identified within the context of four key drivers:
  - Cost reduction,
  - Time reduction,
  - Quality improvement, and
  - Personnel development and empowerment.
  - Goals may be defined at the business level or for a specific component of the business.



Software Re-engineering

Process identification:

- Processes that are critical to achieving the goals defined in the business definition are identified.
- They may then be ranked by importance, by need for change, or in any other way that is appropriate for the reengineering activity.

Process evaluation:

- The existing process is thoroughly analyzed and measured.
- Process tasks are identified; the costs and time consumed by process tasks are noted; and quality/performance problems are isolated.

Process specification and design.

- Based on information obtained during the first three BPR activities, use cases are prepared for each process that is to be redesigned.
- Within the context of BPR, use cases identify a scenario that delivers some outcome to a customer. With the use case as the specification of the process, a new set of tasks are designed for the process.

Software Re-engineering

Prototyping:

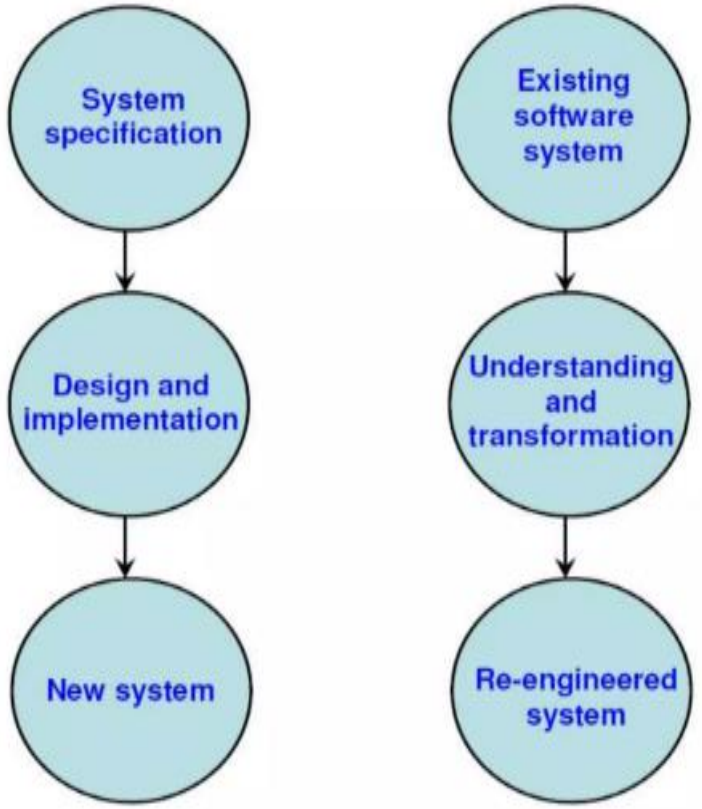
- A redesigned business process must be prototyped before it is fully integrated into the business. This activity “tests” the process so that refinements can be made.

Refinement and instantiation.

- Based on feedback from the prototype, the business process is refined and then instantiated within a business system.

Software Re-engineering

- The principles of re-engineering when applied to software development processes, it is called software reengineering.
- A process of software development which is done to improve the maintainability of a software system. This process includes developing additional features on the software and adding functionalities for better and more efficient software.
- Software re-engineering is concerned with taking existing legacy systems and re- implementing them to make them more maintainable.
- This process encompasses a combination of sub-processes like reverse engineering, forward engineering, reconstructing etc.
- The critical distinction between re-engineering and new software development is the starting point for the development.



Software Re-engineering - The need for software Re-engineering:

VIT-AP UNIVERSITY

The modernization process will be an appropriate option in the following cases:

- When the programming language or platform is no longer supported.
- There is a drastic change in technology, as the market is constantly evolving and, if the company wants to keep up with technology, reengineering process becomes a necessity
- Business processes in the company are changing.
- If the software is initially poor. This sometimes happens because of an attempt of excessive saving at development or opting for amateur performers. If the solution used limits the company's performance and does not work well enough, it influences business processes and reengineering might help to achieve a higher-quality product;
- A technology appears which is perfect for your goals. For example, a completely new market can appear, as happened when the first iPhone came out. This does not happen often nowadays, but still, it may well do.

Software Re-engineering

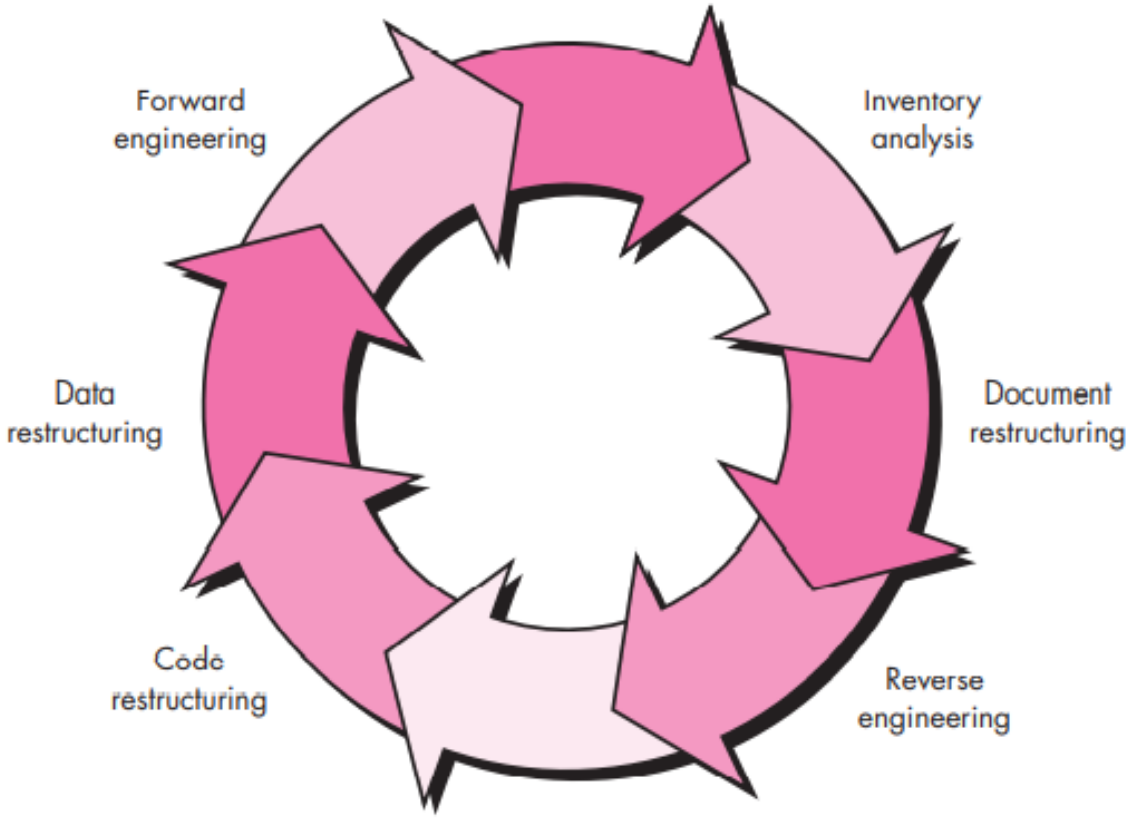
Objectives of Re-engineering:

- To describe a cost-effective option for system evolution.
- To describe the activities involved in the software maintenance process.
- To distinguish between software and data re-engineering and to explain the problems of data re-engineering.
- The Software Reengineering process basically undergoes three main phases. These are
    - Reverse engineering
    - Restructuring, and
    - Forward engineering.

Software Re-engineering

Steps involved in Re-engineering:

1.Inventory Analysis
2.Document Restructuring
3.Reverse Engineering
4.Code Restructuring
5.Data Restructuring
6.Forward Engineering



A software reengineering process model

Software Re-engineering

1. **Inventory Analysis:**
   - Every software organization should have an inventory of all the applications.
   - Inventory can be nothing more than a spreadsheet model containing information that provides a detailed description of every active application.
   - By sorting this information according to business criticality, longevity, current maintainability, and other local important criteria, candidates for re-engineering appear.
   - The resource can then be allocated to a candidate application for re-engineering work.
2. **Document reconstructing:**
   - Documentation of a system either explains how it operates or how to use it.
   - Documentation must be updated.
   - It may not be necessary to fully document an application.
   - The system is business-critical and must be fully re-documented.

Software Re-engineering

VIT-AP UNIVERSITY

3. Reverse Engineering:

- The process of analyzing a program in an effort to create a representation of the program at a higher level of abstraction than source code
- It is a process of design recovery.
- These tools extract data and architectural and procedural design information from an existing program.

4. Code Reconstructing:

- The source code is analyzed using a reconstructing tool.
- Violations of structured programming construct are noted and code is then reconstructed or even rewritten in a more modern programming language .
- The resultant restructured code is reviewed and tested to ensure that no anomalies have been introduced.

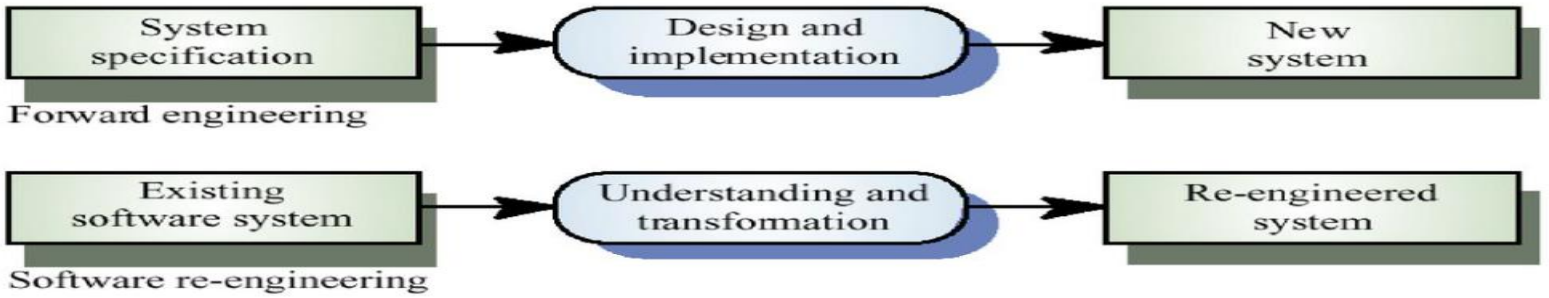
Software Re-engineering

5. Data Restructuring:

- Begins with a reverse engineering activity and the current data architecture is dissected, and the necessary data models are defined.
- Data objects and attributes are identified, and existing data structures are reviewed for quality.

6. Forward Engineering:

- Forward Engineering also called Renovation or Reclamation.
- Not only recovers design information from existing software but uses this information to alter or reconstitute the existing system in an effort to improve its overall quality.
- In most cases, reengineered software reimplements the function of the existing system and also adds new functions and/or improves overall performance



Software Re-engineering

Re-engineering Cost Factors:

- The quality of the software to be re-engineered
- The tool support available for re-engineering
- The extent of the required data conversion
- The availability of expert staff for re-engineering

Advantages of Re-engineering:

- **Reduced Risk:** As the software is already existing, the risk is less as compared to new software development. Development problems, staffing problems and specification problems are the lots of problems which may arise in new software development.
- **Reduced Cost:** The cost of re-engineering is less than the costs of developing new software.
- **Revelation of Business Rules:** As a system is re-engineered, business rules that are embedded in the system are rediscovered.
- **Better use of Existing Staff:** Existing staff expertise can be maintained and extended accommodate new skills during re-engineering.
- **Productivity increase.** By optimizing the code and database the speed of work is increased;

Software Re-engineering

Disadvantages of Re-engineering:

- Practical limits to the extent of re-engineering.
- Major architectural changes or radical reorganizing of the systems data management has to be done manually.
- Re-engineered system is not likely to be as maintainable as a new system developed using modern software Re-engineering methods.

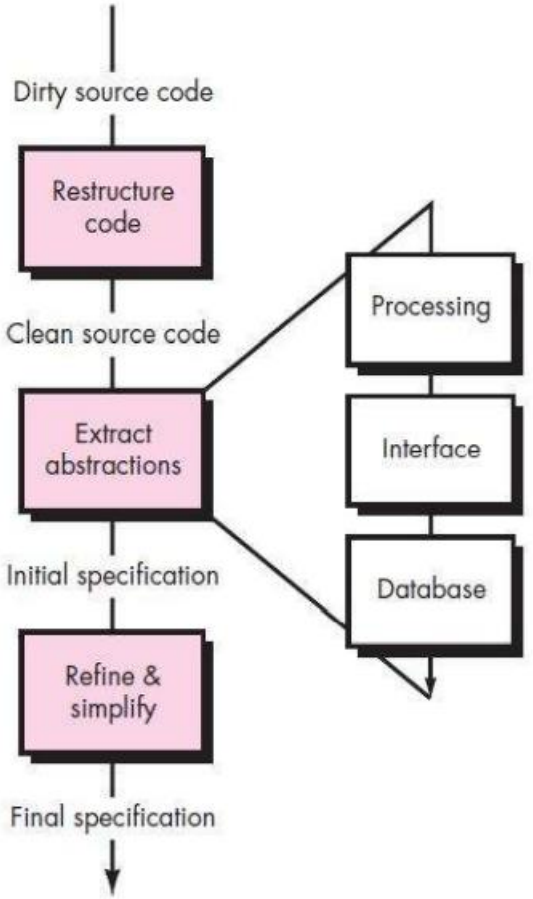
Reverse Engineering

VIT-AP UNIVERSITY

- Reverse engineering is the process followed in order to find difficult, unknown and hidden information about a software system.
- Software reverse engineering is the **process of recovering the design and the requirements specification of a product** from an analysis of its code.
- The **purpose** of reverse engineering is to **facilitate maintenance work by improving the understandability of a system and to produce the necessary documents for a legacy system.**
- Helps to
  - Understand data (internal data structure and database structure)
  - Understand processing (overall functionality of the entire application)
  - Understand user interfaces (basic actions and behavioral response of the system to these actions)

Reverse Engineering

- Before reverse engineering activities can commence, unstructured ("dirty") source code is restructured so that it contains only the structured programming constructs, which makes the source code easier to read and provides the basis for all the subsequent reverse engineering activities.
- The core of reverse engineering is an activity called **extract abstractions**.
- Evaluate the old program and from the (often undocumented) source code, develop a meaningful specification of the processing that is performed, the user interface that is applied, and the program data structures or database that is used.



Reverse engineering process