Unit-5

Software Maintenance

Software maintenance is a part of the Software Development Life Cycle. Its 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.

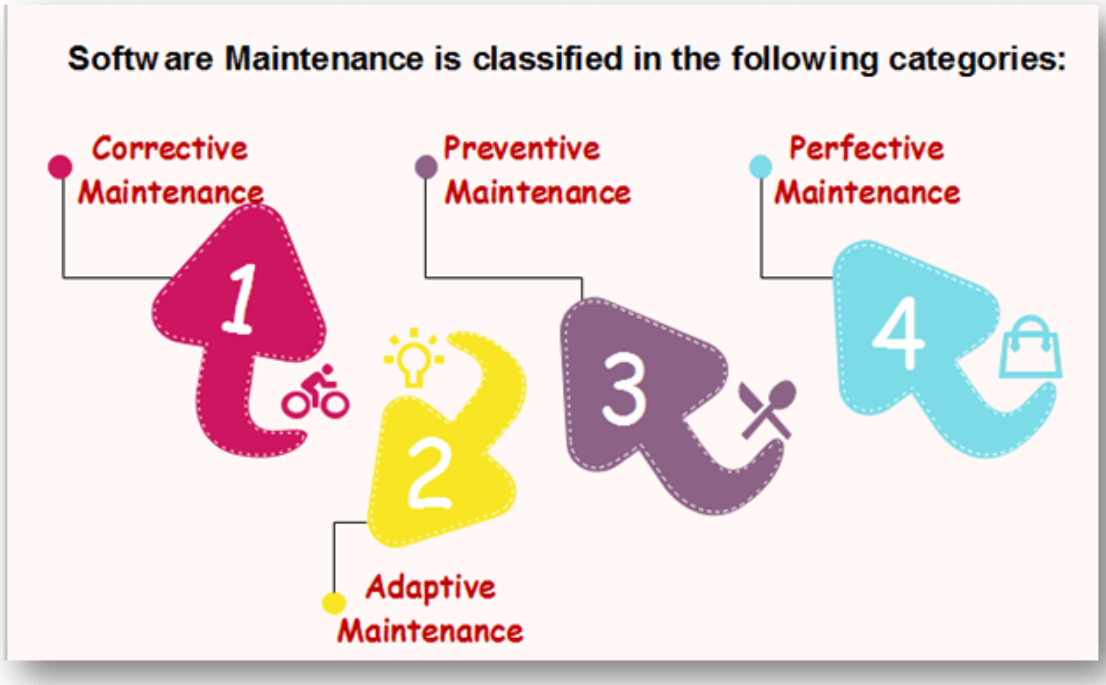
Need for Maintenance

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.

Types of Software Maintenance



1. Corrective Maintenance

Corrective maintenance aims to correct any remaining errors regardless of where they may cause specifications, design, coding, testing, and documentation, etc.

2. Adaptive Maintenance

It contains modifying the software to match changes in the ever-changing environment.

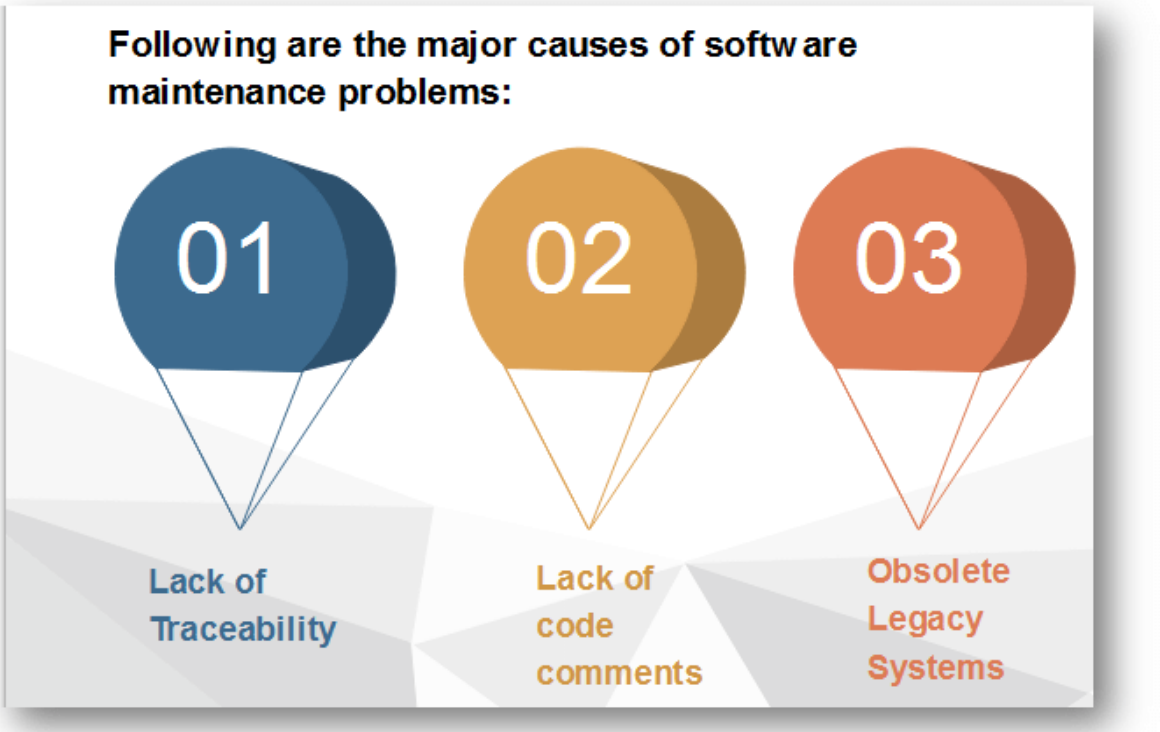
3. Preventive Maintenance

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 old technology is re-engineered using new technology. This maintenance prevents the system from dying out.

4. Perfective Maintenance

It defines improving processing efficiency or performance or restricting the software to enhance changeability. This may contain enhancement of existing system functionality, improvement in computational efficiency, etc.

Causes of Software Maintenance Problems



Lack of Traceability

- Codes are rarely traceable to the requirements and design specifications.
- It makes it very difficult for a programmer to detect and correct a critical defect affecting customer operations.
- Like a detective, the programmer pores over the program looking for clues.
- Life Cycle documents are not always produced even as part of a development project.

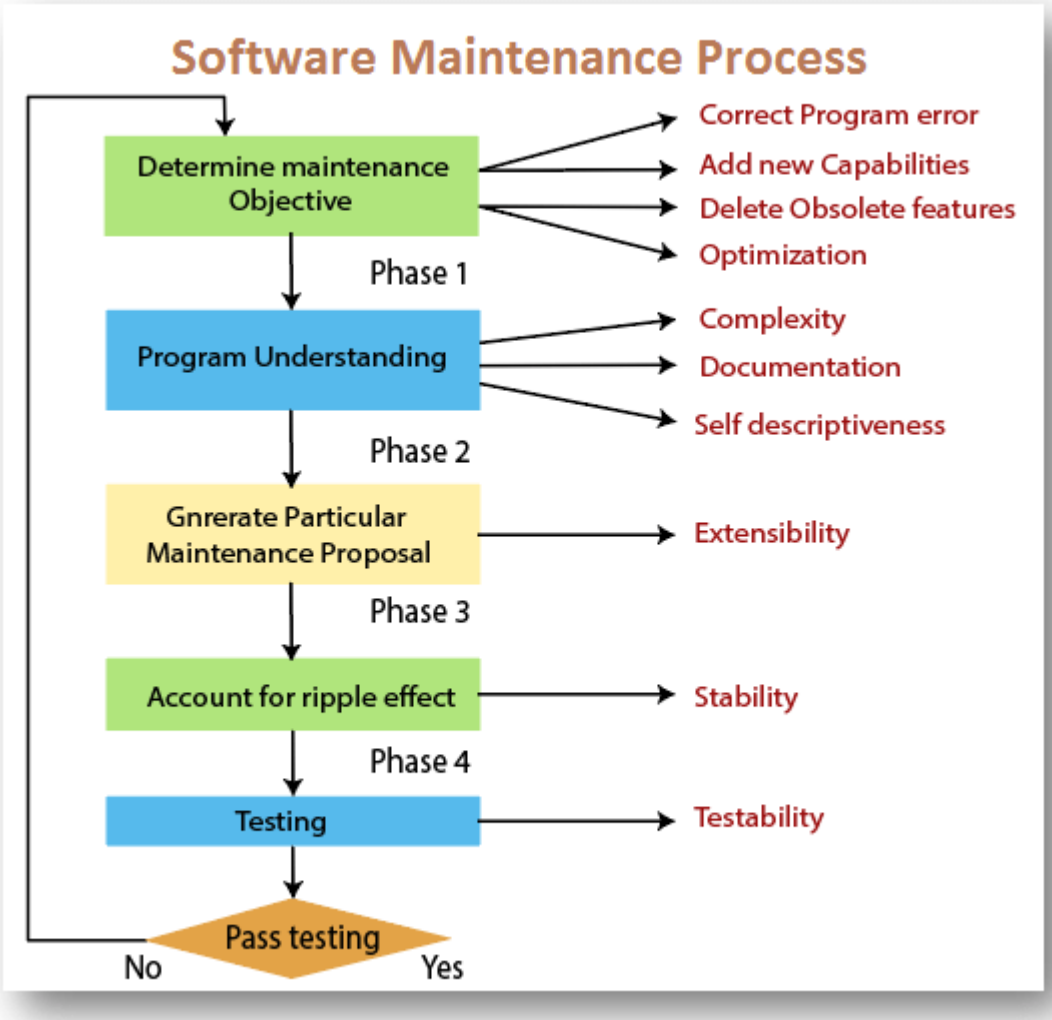
Lack of Code Comments

- Most of the software system codes lack adequate comments. Lesser comments may not be helpful in certain situations.

Obsolete Legacy Systems

- In most of the countries worldwide, the legacy system that provides the backbone of the nation's critical industries, e.g., telecommunications, medical, transportation utility services, were not designed with maintenance in mind.
- They were not expected to last for a quarter of a century or more!
- As a consequence, the code supporting these systems is devoid of traceability to the requirements, compliance to design and programming standards and often includes dead, extra and uncommented code, which all make the maintenance task next to the impossible.

Software Maintenance Process



Program Understanding

The first step consists of analyzing the program to understand.

Generating a Particular maintenance problem

The second phase consists of creating a particular maintenance proposal to accomplish the implementation of the maintenance goals.

Ripple Effect

The third step consists of accounting for all of the ripple effects as a consequence of program modifications.

Modified Program Testing

The fourth step consists of testing the modified program to ensure that the revised application has at least the same reliability level as prior.

Maintainability

Each of these four steps and their associated software quality attributes is critical to the maintenance process. All of these methods must be combined to form maintainability.

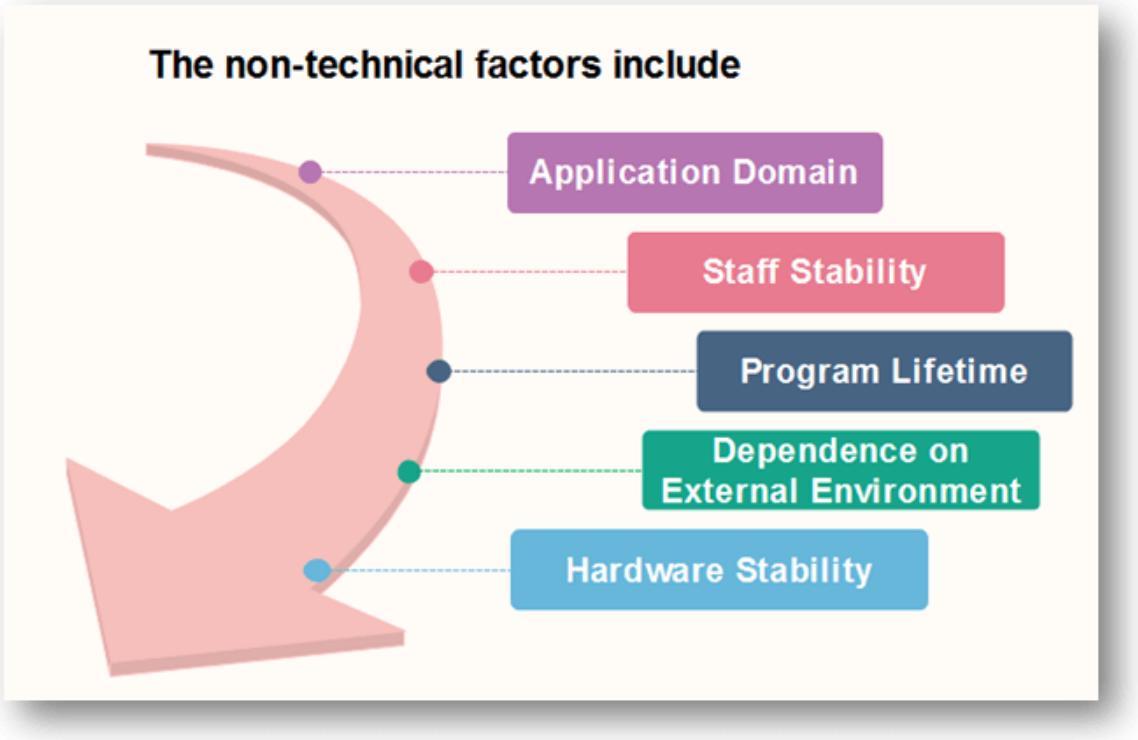
Software Maintenance Cost Factors

There are two types of cost factors involved in software maintenance.

These are

- Non-Technical Factors
- Technical Factors

Non-Technical Factors



1. Application Domain

- If the application of the program is defined and well understood, the system requirements may be definitive and maintenance due to changing needs minimized.
- If the form is entirely new, it is likely that the initial conditions will be modified frequently, as user gain experience with the system.

2. Staff Stability

- It is simple for the original writer of a program to understand and change an application rather than some other person who must understand the program by the study of the reports and code listing.
- If the implementation of a system also maintains that systems, maintenance costs will reduce.
- In practice, the feature of the programming profession is such that persons change jobs regularly. It is unusual for one user to develop and maintain an application throughout its useful life.

3. Program Lifetime

- Programs become obsolete when the program becomes obsolete, or their original hardware is replaced, and conversion costs exceed rewriting costs.

4. Dependence on External Environment

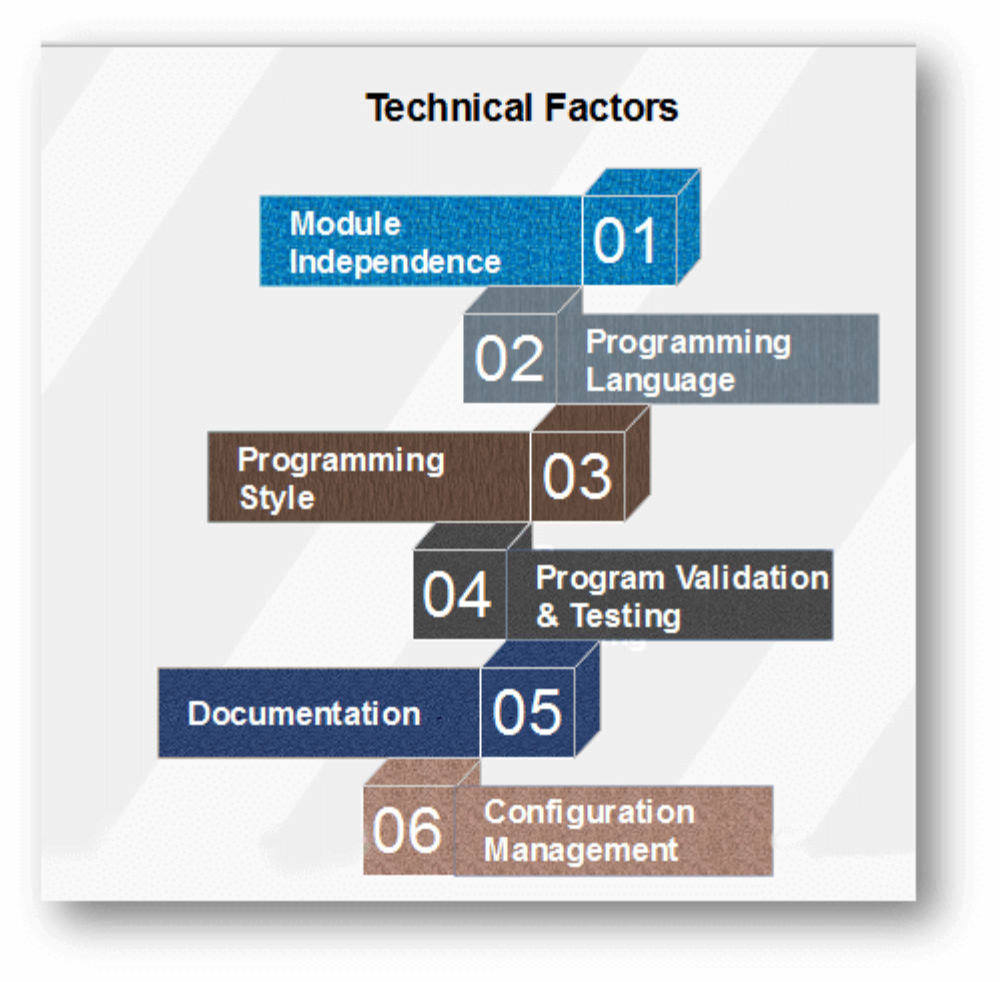
- If an application is dependent on its external environment, it must be modified as the climate changes.
- For example:
- Changes in a taxation system might need payroll, accounting, and stock control programs to be modified.
- Taxation changes are nearly frequent, and maintenance costs for these programs are associated with the frequency of these changes.
- A program used in mathematical applications does not typically depend on humans changing the assumptions on which the program is based.

5. Hardware Stability

- If an application is designed to operate on a specific hardware configuration and that configuration does not changes during the program's lifetime, no maintenance costs due to hardware changes will be incurred.
- Hardware developments are so increased that this situation is rare.
- The application must be changed to use new hardware that replaces obsolete equipment.

Technical Factors

Technical Factors include the following:



Module Independence

It should be possible to change one program unit of a system without affecting any other unit.

Programming Language

Programs written in a high-level programming language are generally easier to understand than programs written in a low-level language.

Programming Style

The method in which a program is written contributes to its understandability and hence, the ease with which it can be modified.

Program Validation and Testing

- Generally, more the time and effort are spent on design validation and program testing, the fewer bugs in the program and, consequently, maintenance costs resulting from bugs correction are lower.

- Maintenance costs due to bug's correction are governed by the type of fault to be repaired.
- Coding errors are generally relatively cheap to correct, design errors are more expensive as they may include the rewriting of one or more program units.
- Bugs in the software requirements are usually the most expensive to correct because of the drastic design which is generally involved.

Documentation

- If a program is supported by clear, complete yet concise documentation, the functions of understanding the application can be associatively straight-forward.
- Program maintenance costs tends to be less for well-reported systems than for the system supplied with inadequate or incomplete documentation.

Configuration Management Techniques

- One of the essential costs of maintenance is keeping track of all system documents and ensuring that these are kept consistent.
- Effective configuration management can help control these costs.

Software Supportability

What is Software Supportability?

Software Supportability is the capability of supporting a software system over its whole product life. This implies satisfying any necessary needs or requirements, but also the provision of equipment, support infrastructure, additional software, facilities, manpower, or any other resource required to maintain the software operational and capable of satisfying its function.

What does Software Support encompass?

Software Support covers the whole software life-cycle once it enters into service. In particular, it covers the following key aspects associated to the software:

- Operation
- Logistics Management
- Modification

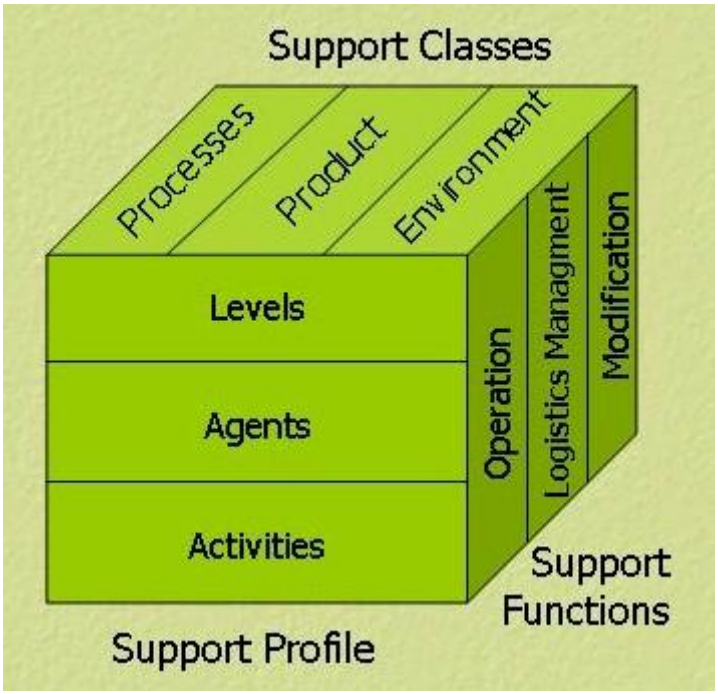
Operation covers all aspects associated to the actual use of the software, including the installation, loading (or unloading), configuration, error recovery and execution of the software.

Logistics Management covers all aspects related to the handling of the software once a new baseline has been produced, until its delivery to the end user.

Modification (often mistakenly called Software Maintenance) covers all aspects related to the evolution of the software due to the need of fixing bugs, or adding/changing functionality due to changing user needs.

These three aspects represent the *functions* associated to software support. Each of this functions has however a set of support classes that characterize it: Modification, for example, has a different software product (a design) than operation (an executable). The environment (a software house vs the user site) is also different. And, of course, things are being done differently for each function, even if the overall process is the same.

On the other hand, all these things are done at different places, by different people, who do different things. This is called the support profile. Combined with the support classes and the support functions, the forms the overall **Support Concept**. A very good description of what a support concept is can be found in SAE JA1006, Software Support Concept.



Software Support Concept

Software Engineering | Re-engineering

Software Re-engineering is a process of software development which is done to improve the maintainability of a software system. Re-engineering is the examination and alteration of a system to reconstitute it in a new form. This process encompasses a combination of sub-processes like reverse engineering, forward engineering, reconstructing etc.

Re-engineering is the reorganizing and modifying existing software systems to mak e them more maintainable.

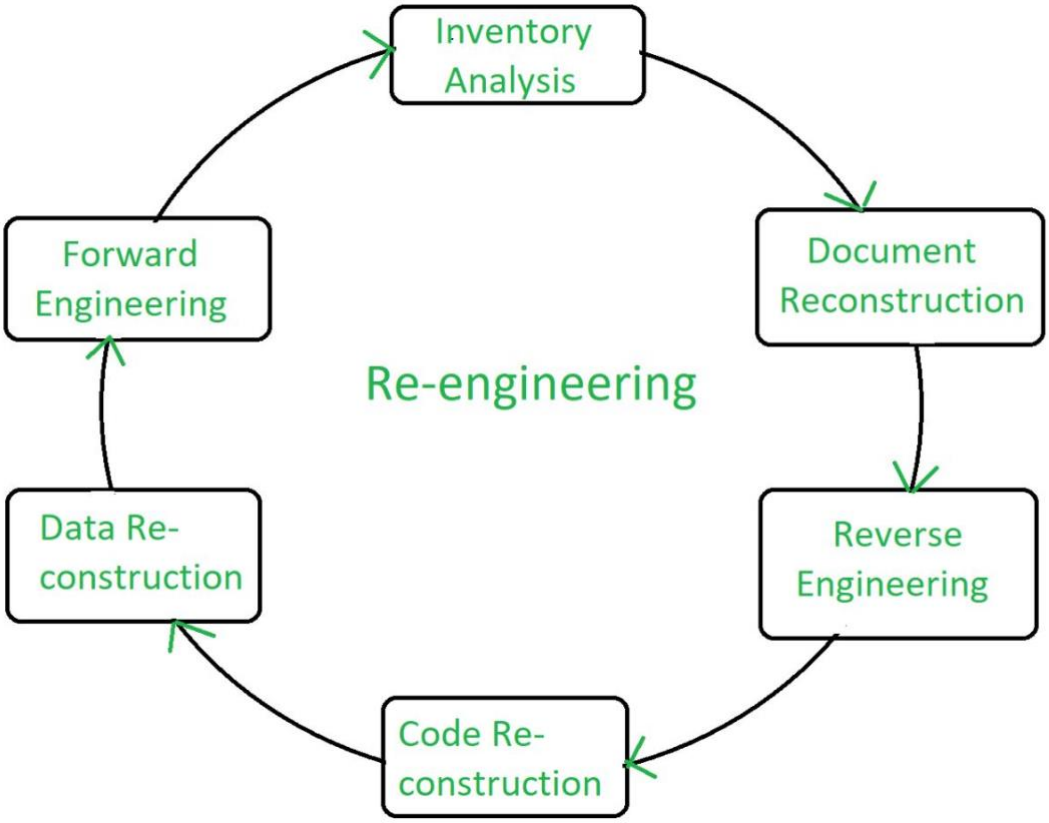
Objectives of Re-engineering:

- 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.

Steps involved in Re-engineering:

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

Diagrammatic Representation:



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.

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.

Software Re-Engineering is the examination and alteration of a system to reconstitute it in a new form. The principles of Re-Engineering when applied to the software development process is called software re-engineering. It affects positively at software cost, quality, service to the customer and speed of delivery. In Software Re-engineering, we are improving the software to make it more efficient and effective.

The need of software Re-engineering: Software re-engineering is an economical process for software development and quality enhancement of the product. This process enables us to identify the useless consumption of deployed resources and the constraints that are restricting the development process so that the development process could be made easier and cost-effective (time, financial, direct advantage, optimize the code, indirect benefits, etc.) and maintainable. The software reengineering is necessary for having-

a) Boost up productivity: Software reengineering increase productivity by optimizing the code and database so that processing gets faster.

b) Processes in continuity: The functionality of older software product can be still used while the testing or development of software.

c) Improvement opportunity: Meanwhile the process of software reengineering, not only software qualities, features and functionality but also your skills are refined, new ideas hit in your mind. This makes the developers mind accustomed to capturing new opportunities so that more and more new features can be developed.

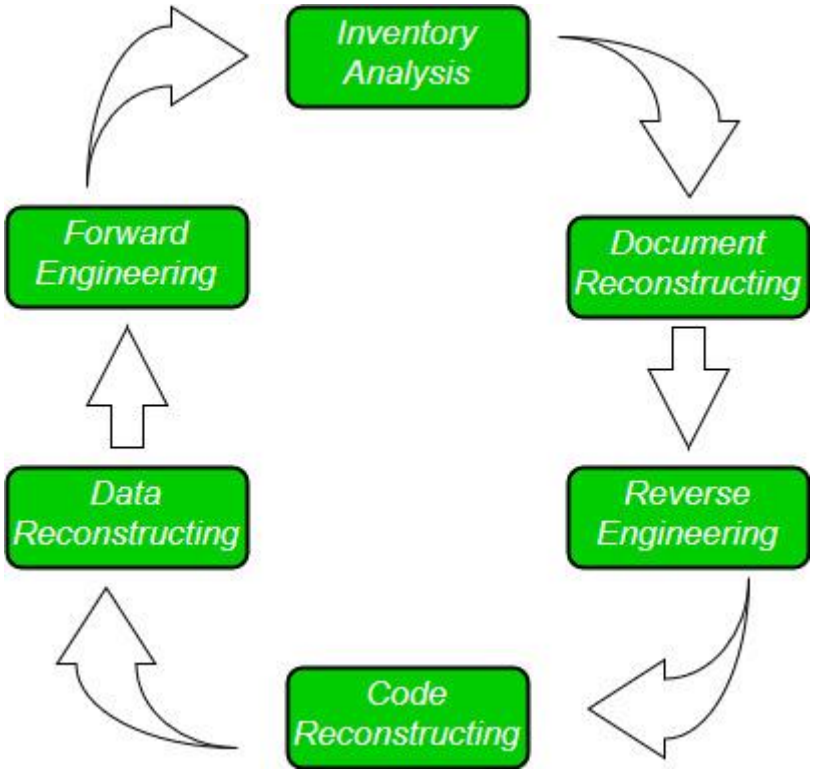
d) Reduction in risks: Instead of developing the software product from scratch or from the beginning stage here developers develop the product from its existing stage to enhance some specific features that are brought in concern by stakeholders or its users. Such kind of practice reduces the chances of fault fallibility.

e) Saves time: As we stated above here that the product is developed from the existing stage rather than the beginning stage so the time consumes in software engineering is lesser.

f) <u>Optimization</u>: This process refines the system features, functionalities and reduces the complexity of the product by consistent optimization as maximum as possible.

Re-Engineering cost factors:

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



Software Re-Engineering Activities:

1. Inventory Analysis:

Every software organisation 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.

3. Reverse Engineering:
Reverse engineering is a process of design recovery. Reverse engineering tools extract data, architectural and procedural design information from an existing program.

4. Code Reconstructing:

- To accomplish code reconstructing, the source code is analysed using a reconstructing tool. Violations of structured programming construct are noted and code is then reconstructed.
- The resultant restructured code is reviewed and tested to ensure that no anomalies have been introduced.

5. Data Restructuring:

- Data restructuring begins with a reverse engineering activity.
- Current data architecture is dissected, and the necessary data models are defined.
- Data objects and attributes are identified, and existing data structure are reviewed for quality.

6. Forward Engineering:
Forward Engineering also called as renovation or reclamation not only for 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.

Reverse Engineering

Reverse Engineering –

Reverse Engineering is processes of extracting knowledge or design information from anything man-made and reproducing it based on extracted information. It is also called back Engineering.

Software Reverse Engineering –

Software Reverse Engineering is the process of recovering the design and the requirements specification of a product from an analysis of it's code. Reverse Engineering is becoming important, since several existing software products, lack proper documentation, are highly unstructured, or their structure has degraded through a series of maintenance efforts.

Why Reverse Engineering?

- Providing proper system documentation.
- Recovery of lost information.
- Assisting with maintenance.
- Facility of software reuse.
- Discovering unexpected flaws or faults.

Used of Software Reverse Engineering –

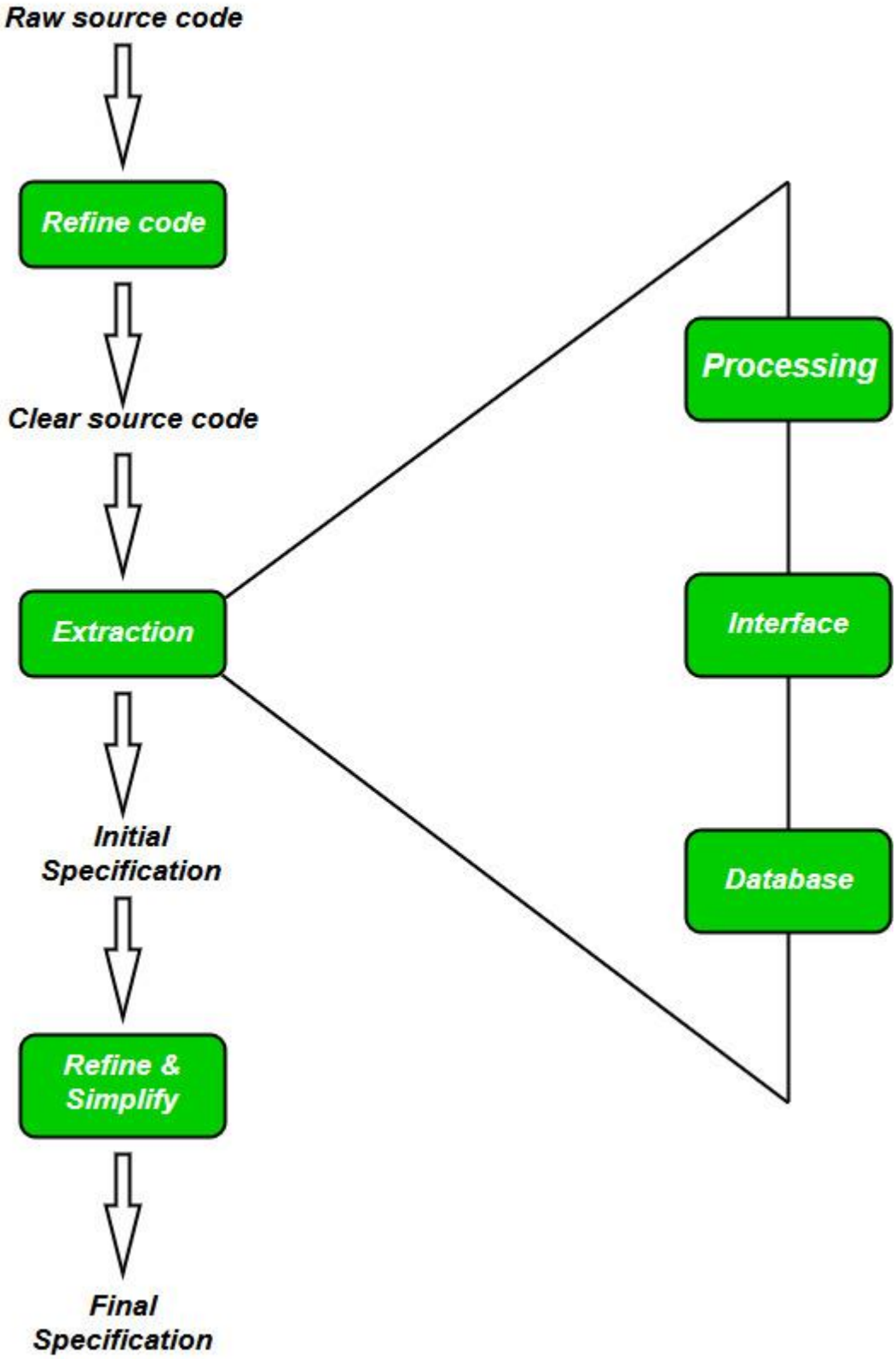
- Software Reverse Engineering is used in software design, reverse engineering enables the developer or programmer to add new features to the existing software with or without knowing the source code.
- Reverse engineering is also useful in software testing, it helps the testers to study the virus and other malware code .

Software Engineering | Reverse Engineering

Software Reverse Engineering is a process of recovering the design, requirement specifications and functions of a product from an analysis of its code. It builds a program database and generates information from this.
The purpose of reverse engineering is to facilitate the maintenance work by improving the understandability of a system and to produce the necessary documents for a legacy system.

Reverse Engineering Goals:

- Cope with Complexity.
- Recover lost information.
- Detect side effects.
- Synthesise higher abstraction.
- Facilitate Reuse.



Steps of Software Reverse Engineering:

1. **Collection Information:**
   This step focuses on collecting all possible information (i.e., source design documents etc.) about the software.

2. **Examining the information:**
   The information collected in step-1 as studied so as to get familiar with the system.

3. **Extracting the structure:**
   This step concerns with identification of program structure in the form of structure chart where each node corresponds to some routine.

4. **Recording the functionality:**
   During this step processing details of each module of the structure, charts are recorded using structured language like decision table, etc.

5. **Recording data flow:**
   From the information exA tracted in step-3 and step-4, set of data flow diagrams are derived to show the flow of data among the processes.

6. **Recording control flow:**
   High level control structure of the software is recorded.

7. **Review extracted design:**
   Design document extracted is reviewed several times to ensure consistency and correctness. It also ensures that the design represents the program.

8. **Generate documentation:**
   Finally, in this step, the complete documentation including SRS, design document, history, overview, etc. are recorded for future use.

Reverse Engineering Tools:
Reverse engineering if done manually would consume lot of time and human labour and hence must be supported by automated tools. Some of tools are given below:

- **CIAO and CIA:** A graphical navigator for software and web repositories along with a collection of Reverse Engineering tools.
- **Rigi:** A visual software understanding tool.
- **Bunch:** A software clustering/modularization tool.

- **GEN++:** An application generator to support development of analysis tools for the C++ language.
- **PBS:** Software Bookshelf tools for extracting and visualizing the architecture of programs.

Difference between Forward Engineering and Reverse Engineering

- Last Updated : 18 Aug, 2021

Forward Engineering:
Forward Engineering is a method of creating or making an application with the help of the given requirements. Forward engineering is also known as Renovation and Reclamation. Forward engineering is required high proficiency skills. It takes more time to construct or develop an application. Forward engineering is a technique of creating high-level models or designs to make in complexities and low-level information. Therefore this kind of engineering has completely different principles in numerous package and information processes. Forward Engineering applies of all the software engineering process which contains SDLC to recreate associate existing application. It is near to full fill new needs of the users into re-engineering.

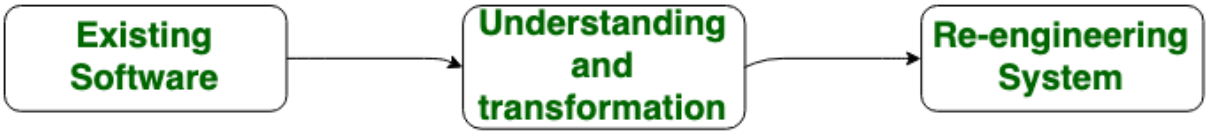
Characteristics of forward engineering:

1. Forward engineering is a variety of engineering that has different principles in numerous package and information processes.
2. Forward engineering is vital in IT as a result of it represents the 'normal' development process.
3. Forward engineering deals with the conversion of business processes, services, and functions into applications.
4. In this method business model is developed first. Then, a top-to-down approach is followed to urge the package from the model developed.
5. Forward engineering tools are accustomed move from implementation styles and logic to the event of supply code.
6. It essentially permits the user to develop a business model which may then be translated into data system components.
7. These tools basically follow the top-to down approach. System creator and visual Analyst is a forward engineering CASE tool.

Forward Engineering

Reverse Engineering:

Reverse Engineering is also known as backward engineering, is the process of forward engineering in reverse. In this, the information is collected from the given or existing application. It takes less time than forward engineering to develop an application. In reverse engineering, the application is broken to extract knowledge or its architecture.



Reverse Engineering

Difference between Forward Engineering and Reverse Engineering:

S.NO	Forward Engineering	Reverse Engineering
1.	In forward engineering, the application are developed with the given requirements.	In reverse engineering or backward engineering, the information are collected from the given application.
2.	Forward Engineering is a high proficiency skill.	Reverse Engineering or backward engineering is a low proficiency skill.
3.	Forward Engineering takes more time to develop an application.	While Reverse Engineering or backward engineering takes less time to develop an application.
4.	The nature of forward engineering is Prescriptive.	The nature of reverse engineering or backward engineering is Adaptive.
5.	In forward engineering, production is started with given requirements.	In reverse engineering, production is started by taking the products existing products.

What is Risk?

"Tomorrow problems are today's risk." Hence, a clear definition of a "risk" is a problem that could cause some loss or threaten the progress of the project, but which has not happened yet.

These potential issues might harm cost, schedule or technical success of the project and the quality of our software device, or project team morale.

Risk Management is the system of identifying addressing and eliminating these problems before they can damage the project.

We need to differentiate risks, as potential issues, from the current problems of the project.

Different methods are required to address these two kinds of issues.

For example, staff storage, because we have not been able to select people with the right technical skills is a current problem, but the threat of our technical persons being hired away by the competition is a risk.

Risk Management

A software project can be concerned with a large variety of risks. In order to be adept to systematically identify the significant risks which might affect a software project, it is essential to classify risks into different classes. The project manager can then check which risks from each class are relevant to the project.

There are three main classifications of risks which can affect a software project:

1. Project risks
2. Technical risks
3. Business risks

1. Project risks: Project risks concern differ forms of budgetary, schedule, personnel, resource, and customer-related problems. A vital project risk is schedule slippage. Since the software is intangible, it is very tough to monitor and control a software project. It is very tough to control something which cannot be identified. For any manufacturing program, such as the manufacturing of cars, the plan executive can recognize the product taking shape.

2. Technical risks: Technical risks concern potential method, implementation, interfacing, testing, and maintenance issue. It also consists of an ambiguous specification, incomplete specification, changing specification, technical uncertainty, and technical obsolescence. Most technical risks appear due to the development team's insufficient knowledge about the project.

3. Business risks: This type of risks contain risks of building an excellent product that no one need, losing budgetary or personnel commitments, etc.

Other risk categories

1. Known risks: Those risks that can be uncovered after careful assessment of the project program, the business and technical environment in which the plan is being developed, and more reliable data sources (e.g., unrealistic delivery date)

2. Predictable risks: Those risks that are hypothesized from previous project experience (e.g., past turnover)

3. Unpredictable risks: Those risks that can and do occur, but are extremely tough to identify in advance.

Principle of Risk Management

1. **Global Perspective:** In this, we review the bigger system description, design, and implementation. We look at the chance and the impact the risk is going to have.
2. **Take a forward-looking view:** Consider the threat which may appear in the future and create future plans for directing the next events.
3. **Open Communication:** This is to allow the free flow of communications between the client and the team members so that they have certainty about the risks.
4. **Integrated management:** In this method risk management is made an integral part of project management.
5. **Continuous process:** In this phase, the risks are tracked continuously throughout the risk management paradigm.

Risk Management Activities

Risk management consists of three main activities, as shown in fig:

Risk Assessment

The objective of risk assessment is to division the risks in the condition of their loss, causing potential. For risk assessment, first, every risk should be rated in two methods:

- The possibility of a risk coming true (denoted as r).
- The consequence of the issues relates to that risk (denoted as s).

Based on these two methods, the priority of each risk can be estimated:

$$p = r * s$$

Where p is the priority with which the risk must be controlled, r is the probability of the risk becoming true, and s is the severity of loss caused due to the risk becoming true. If all identified risks are set up, then the most likely and damaging risks can be controlled first, and more comprehensive risk abatement methods can be designed for these risks.

1. Risk Identification: The project organizer needs to anticipate the risk in the project as early as possible so that the impact of risk can be reduced by making effective risk management planning.

A project can be of use by a large variety of risk. To identify the significant risk, this might affect a project. It is necessary to categories into the different risk of classes.

There are different types of risks which can affect a software project:

1. **Technology risks:** Risks that assume from the software or hardware technologies that are used to develop the system.
2. **People risks:** Risks that are connected with the person in the development team.
3. **Organizational risks:** Risks that assume from the organizational environment where the software is being developed.
4. **Tools risks:** Risks that assume from the software tools and other support software used to create the system.
5. **Requirement risks:** Risks that assume from the changes to the customer requirement and the process of managing the requirements change.
6. **Estimation risks:** Risks that assume from the management estimates of the resources required to build the system

2. Risk Analysis: During the risk analysis process, you have to consider every identified risk and make a perception of the probability and seriousness of that risk.

There is no simple way to do this. You have to rely on your perception and experience of previous projects and the problems that arise in them.

It is not possible to make an exact, the numerical estimate of the probability and seriousness of each risk. Instead, you should authorize the risk to one of several bands:

1. The probability of the risk might be determined as very low (0-10%), low (10-25%), moderate (25-50%), high (50-75%) or very high (+75%).
2. The effect of the risk might be determined as catastrophic (threaten the survival of the plan), serious (would cause significant delays), tolerable (delays are within allowed contingency), or insignificant.

Risk Control

It is the process of managing risks to achieve desired outcomes. After all, the identified risks of a plan are determined; the project must be made to include the most harmful and the most likely risks. Different risks need different containment methods. In fact, most risks need ingenuity on the part of the project manager in tackling the risk.

There are three main methods to plan for risk management:

1. **Avoid the risk:** This may take several ways such as discussing with the client to change the requirements to decrease the scope of the work, giving incentives to the engineers to avoid the risk of human resources turnover, etc.
2. **Transfer the risk:** This method involves getting the risky element developed by a third party, buying insurance cover, etc.
3. **Risk reduction:** This means planning method to include the loss due to risk. For instance, if there is a risk that some key personnel might leave, new recruitment can be planned.

Risk Leverage: To choose between the various methods of handling risk, the project plan must consider the amount of controlling the risk and the corresponding reduction of risk. For this, the risk leverage of the various risks can be estimated.

Risk leverage is the variation in risk exposure divided by the amount of reducing the risk.

Risk leverage = (risk exposure before reduction - risk exposure after reduction) / (cost of reduction)

1. Risk planning: The risk planning method considers each of the key risks that have been identified and develop ways to maintain these risks.

For each of the risks, you have to think of the behavior that you may take to minimize the disruption to the plan if the issue identified in the risk occurs.

You also should think about data that you might need to collect while monitoring the plan so that issues can be anticipated.

Again, there is no easy process that can be followed for contingency planning. It rely on the judgment and experience of the project manager.

2. Risk Monitoring: Risk monitoring is the method king that your assumption about the product, process, and business risks has not changed.