Software Maintenance is the process of modifying a software product after it has been delivered to the customer. The main purpose of software maintenance is to modify and update software applications after delivery to correct faults and to improve performance.

9.1 Need for software maintenance

Market Conditions - Policies, which changes over the time, such as taxation and newly introduced constraints like, how to maintain bookkeeping, may trigger need for modification.

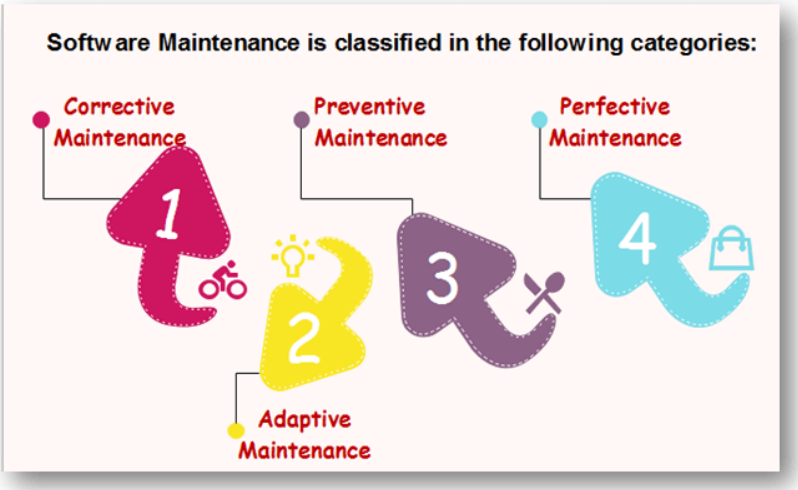
Client Requirements - Over the time, customer may ask for new features or functions in the software.

Host Modifications - If any of the hardware and/or platform (such as operating system) of the target host changes, software changes are needed to keep adaptability.

Organization Changes - If there is any business level change at client end, such as reduction of organization strength, acquiring another company, organization venturing into new business, need to modify in the original software may arise.

9.2 Types of software maintenance

In a software lifetime, type of maintenance may vary based on its nature. It may be just a routine maintenance tasks as some bug discovered by some user or it may be a large event in itself based on maintenance size or nature. Following are some types of maintenance based on their characteristics:



Corrective Maintenance - This includes modifications and updations done in order to correct or fix problems, which are either discovered by user or concluded by user error reports.

Adaptive Maintenance - This includes modifications and updations applied to keep the software product up-to date and tuned to the ever changing world of technology and business environment.

Perfective Maintenance - This includes modifications and updates done in order to keep the software usable over long period of time. It includes new features, new user requirements for refining the software and improve its reliability and performance.

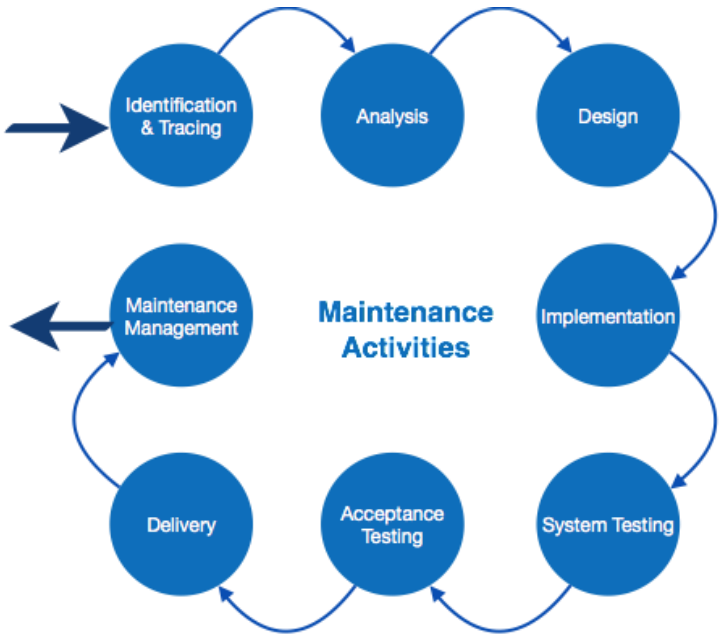
Preventive Maintenance - This includes modifications and updations to prevent future problems of the software. It aims to attend problems, which are not significant at this moment but may cause serious issues in future.

9.3 Software maintenance process model

IEEE provides a framework for sequential maintenance process activities. It can be used in iterative manner and can be extended so that customized items and processes can be included.

Maintenance Activities

These activities go hand-in-hand with each of the following phase:



Identification & Tracing - It involves activities pertaining to identification of requirement of modification or maintenance. It is generated by user or system may itself report via logs or error messages. Here, the maintenance type is classified also.

Analysis - The modification is analyzed for its impact on the system including safety and security implications. If probable impact is severe, alternative solution is looked for. A set of required modifications is then materialized into requirement specifications. The cost of modification/maintenance is analyzed and estimation is concluded.

Design - New modules, which need to be replaced or modified, are designed against requirement specifications set in the previous stage. Test cases are created for validation and verification.

Implementation - The new modules are coded with the help of structured design created in the design step. Every programmer is expected to do unit testing in parallel.

System Testing - Integration testing is done among newly created modules. Integration testing is also carried out between new modules and the system. Finally the system is tested as a whole, following regressive testing procedures.

Acceptance Testing - After testing the system internally, it is tested for acceptance with the help of users. If at this state, user complaints some issues they are addressed or noted to address in next iteration.

Delivery - After acceptance test, the system is deployed all over the organization either by small update package or fresh installation of the system. The final testing takes place at client end after the software is delivered.

Training facility is provided if required, in addition to the hard copy of user manual.

Maintenance management - Configuration management is an essential part of system maintenance. It is aided with version control tools to control versions, semi-version or patch management.

9.4 Software maintenance cost

The factors affecting software maintenance cost are divided into two groups.

Technical

- **Module independence.** The ability to change any of the system's program blocks without affecting the others;
- **Programming language**. Programs that are written in a high-level programming language are easier to understand than those in a low-level language. High-level languages include C ++, C #, Delphi, Fortran, Java, JavaScript, Lisp, Pascal, PHP, Ruby, Python, Perl. Low-level languages include bytecode, IL, assembly language, and CIL;
- **Programming style**. The ease of modifying and understanding a program is determined by the programmer's writing method;
- **Validation and testing the program**. The more time is spent on checking the design and testing the program, the fewer errors are found in the program and the lower the software maintenance cost. Error correction costs are governed by the type of fault. Software requirement errors are the most costly;
- **Documentation**. Maintenance costs will be lower with clear and complete documentation;
- **Configuration management techniques**. One of the costs involved has to do with keeping track of system documents and ensuring their consistency. It means that effective configuration management can help reduce these costs.

Non-Technical

- **Scope**. When a software application is well-defined, system requirements are less likely to change over time, so maintenance due to changing needs is minimal. If the app is completely new, you should be prepared for significant support costs due to frequent changes in conditions as the user gets more experience with the software;
- **Stable team**. When the composition of the software development team changes, new team members will take some time to get the hang of everything. Because of this, making any changes to the software becomes more difficult;
- **Software lifecycle**. When software becomes obsolete, the original hardware is replaced, and the conversion costs exceed the rewrite costs;
- **The external environment**. When software depends on the external environment, modification is necessary at the slightest external change;

- **Stability of hardware**. If the software is executed on a specific hardware configuration that does not change during the entire lifetime of the software, then maintenance costs are reduced to zero. However, this situation is quite rare due to constant hardware development.

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.

Use 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.