Volume 3, 2020 ISSN: 2617-4596
DOI: https://doi.org/10.31058/j.mana.2020.31002

A Review on Importance of Maintenance in Software Engineering

Bindia Tarika[1*]

[1] Computer Science & Engineering, PTU, Punjab, India

Email Address
bindiatarika11@gmail.com (Bindia Tarika)
*Correspondence: bindiatarika11@gmail.com

Received: 21 December 2019**; Accepted:** 20 February 2020**; Published:** 10 March 2020

Abstract:
Software maintenance is widely accepted part of SDLC now a days. It stands for all the modifications and updations done after the delivery of software product. A common perception of maintenance is that it merely involves fixing defects. However, one study indicated that over 80% of maintenance effort is used for non-corrective actions. This perception is perpetuated by users submitting problem reports that in reality are functionality enhancements to the system. This paper describes the importance of maintenance by describing the each phase of maintenance cycle and by describing the different types of maintenance and their needs at different stages of software development.

Keywords:
Software, SDLC, SMLC, Maintenance, Phase

1. Introduction

Software maintenance is a part of the SDLC (Software Development Life Cycle). Its main purpose is to modify and update software application after delivery to correct faults and to improve the performance of the system. It is a very broad activity that takes place soon after the development completed. It optimizes the system's performance by reducing errors, eliminating useless development and applying advanced development.

Software maintenance is a vast activity which includes optimization, error correction, deletion of discarded features and enhancement of existing features. Since these changes are necessary, a mechanism must be created for estimation, controlling and making modifications. The essential part of software maintenance requires preparation of an accurate plan during the development cycle. Typically, maintenance takes up about 40-80% of the project cost, usually closer to the higher pole. Hence, a focus on maintenance definitely helps keep costs down.

Software maintenance sustains the software product throughout its life cycle (from development to operations). Modification requests are logged and tracked, the impact of proposed changes is determined, code and other software artifacts are modified, testing is conducted, and a new version of the software product is released. Also,

training and daily support are provided to users. The term maintainer is defined as an organization that performs maintenance activities.

Software development gets completed within 5 years (depends on the complexity) while software maintenance is an ongoing activity and can be extended up to 15-20 years.

"Software maintenance is the process of modifying a software system or component after delivery to correct faults, improve performance, or adapt to a changed environment" – The Institute of Electrical and Electronics Engineers (IEEE).

Put simply, software maintenance is the process where software vendors provide updates, modifications, bug fixes, patches and additional features to existing software solutions to increase performance. Typically software maintenance fees are a small percentage of overall license fees paid on an annual or monthly basis.

1.1. Key Issues in Software Maintenance

A number of key issues must be dealt with to ensure the effective maintenance of software. Software maintenance provides unique technical and management challenges for software engineers—for example, trying to find a fault in software containing a large number of lines of code that another software engineer developed. Similarly, competing with software developers for resources is a constant battle. Planning for a future release, which often includes coding the next release while sending out emergency patches for the current release, also creates a challenge. The following section presents some of the technical and management issues related to software maintenance. They have been grouped under the following topic headings:Technical issues, Management issues, Cost estimation, and Measurement.

1.1.1. Technical Issues

Limited understanding refers to how quickly a software engineer can understand where to make a change or correction in software that he or she did not develop. Research indicates that about half of the total maintenance effort is devoted to understanding the software to be modified. Thus, the topic of software comprehension is of great interest to software engineers. Comprehension is more difficult in text-oriented representation—in source code, for example—where it is often difficult to trace the evolution of software through its releases/ versions if changes are not documented and if the developers are not available to explain it, which is often the case. Thus, software engineers may initially have a limited understanding of the software; much has to be done to remedy this.

1.1.2. Testing

The cost of repeating full testing on a major piece of software is significant in terms of time and money. In order to ensure that the requested problem reports are valid, the maintainer should replicate or verify problems by running the appropriate tests. Regression testing (the selective retesting of software or a component to verify that the modifications have not caused unintended effects) is an important testing concept in maintenance. Additionally, finding time to test is often difficult. Coordinating tests when different members of the maintenance team are working on different problems at the same time remains a challenge. When software performs critical functions, it may be difficult to bring it offline to test. Tests cannot be executed in the most meaningful place–the production system. The Software Testing KA provides

additional information and references on this matter in its subtopic on regression testing.

1.1.3. Impact Analysis

Impact analysis describes how to conduct, cost effectively, a complete analysis of the impact of a change in existing software. Maintainers must possess an intimate knowledge of the software's structure and content. They use that knowledge to perform impact analysis, which identifies all systems and software products affected by a software change request and develops an estimate of the resources needed to accomplish the change. Additionally, the risk of making the change is determined. The change request, sometimes called a modification request (MR) and often called a problem report (PR), must first be analyzed and translated into software terms. Impact analysis is performed after a change request enters the software configuration management process.

The severity of a problem is often used to decide how and when it will be fixed. The software engineer then identifies the affected components. Several potential solutions are provided, followed by a recommendation as to the best course of action.

Software designed with maintainability in mind greatly facilitates impact analysis. More information can be found in the Software Configuration Management KA.

1.1.4. Maintainability

Maintainability is defined as the capability of the software product to be modified. Modifications may include corrections, improvements, or adaptation of the software to changes in environment as well as changes in requirements and functional specifications.

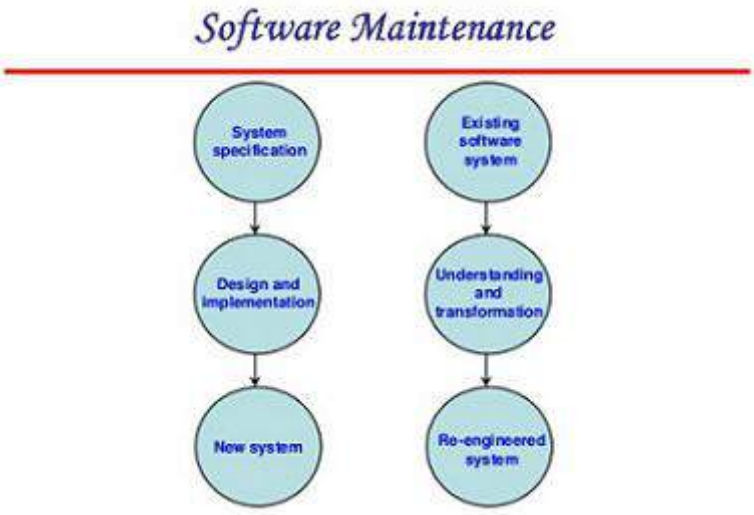

Figure 1. *Software Maintenance.*

As a primary software quality characteristic, maintainability should be specified, reviewed, and controlled during software development activities in order to reduce maintenance costs. When done successfully, the software's maintainability will improve. Maintainability is often difficult to achieve because the subcharacteristics are often not an important focus during the process of software development. The developers are, typically, more preoccupied with many other activities and frequently prone to disregard the maintainer's requirements. This in turn can, and often does, result in a lack of software documentation and test environments, which is a leading

cause of difficulties in program comprehension and subsequent impact analysis. The presence of systematic and mature processes, techniques, and tools helps to enhance the maintainability of software.

2. Software Maintenance Life Cycle

Changes are implemented in the software system by following a software maintenance process, which is known as Software Maintenance Life Cycle (SMLC).This life cycle comprises seven phases, namely, problem identification, analysis, design, implementation, system testing, acceptance testing, and delivery phase.

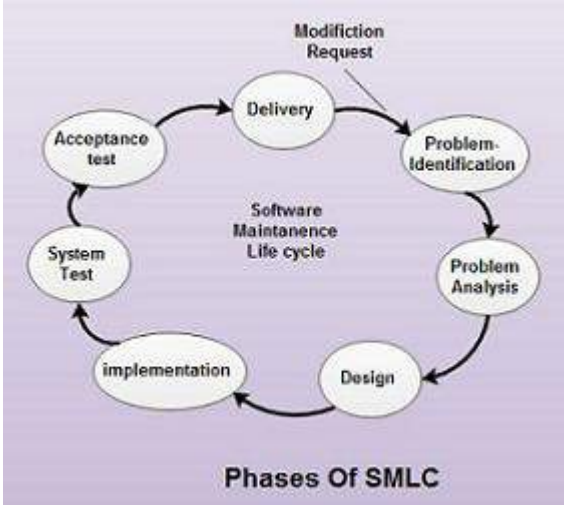

Figure 2. *Phases of SMLC.*

2.1. Problem Identification Phase

In this phase, the requests for modifications in the software are identified and assigned an identification number. Each Modification Request (MR) is then assessed to determine to which type of maintenance activity (corrective, adaptive, perfective, and preventive) the MR belongs. After classification, each MR is assigned with a priority to determine the order in which it is to be processed.

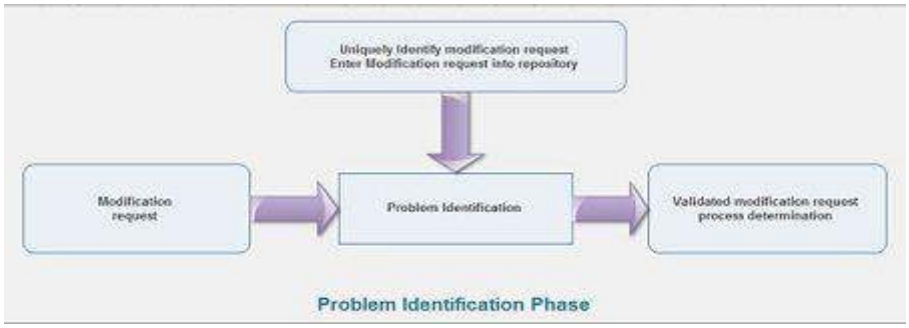

Figure 3. *Problem Identification.*

2.2. Problem Analysis Phase

In this phase, the feasibility and scope of each validated modification request are determined and a plan is prepared to incorporate the changes in the software. The input attribute comprises validated modification request, initial estimate of resources, project documentation, and repository information.

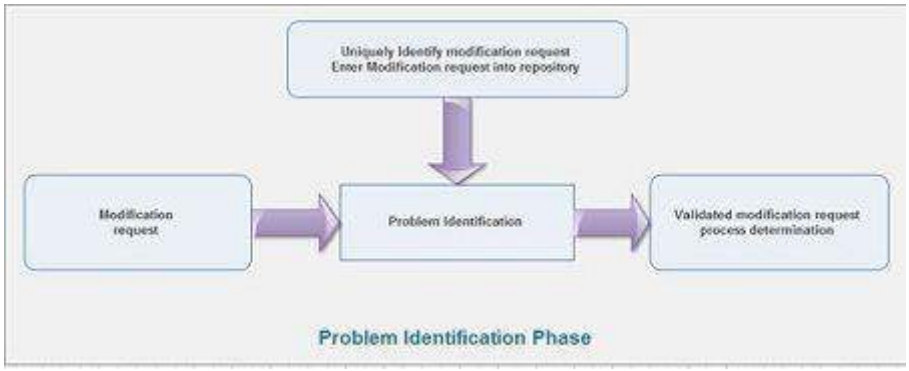

Figure 4. *Problem Analysis.*

2.3. Design Phase

In this phase, the modifications to be made in the software are designed. The input attribute comprises outputs produced by analysis phase (detailed analysis), project and system documentation, software's source code, and databases.

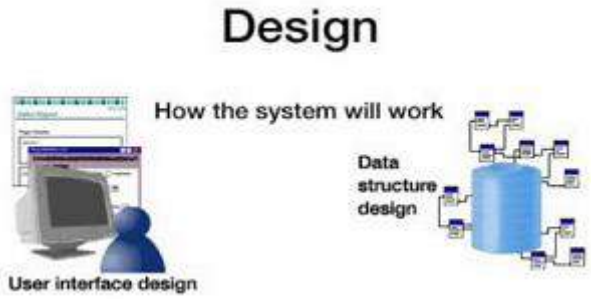

Figure 5. *Design.*

2.4. Implementation Phase

In this phase, the actual modifications in the software code are made, new features that support the specification of present software are added, and the modified software is installed. The input attribute comprises the source code, the output of design phase, and the modified system and project documentation.

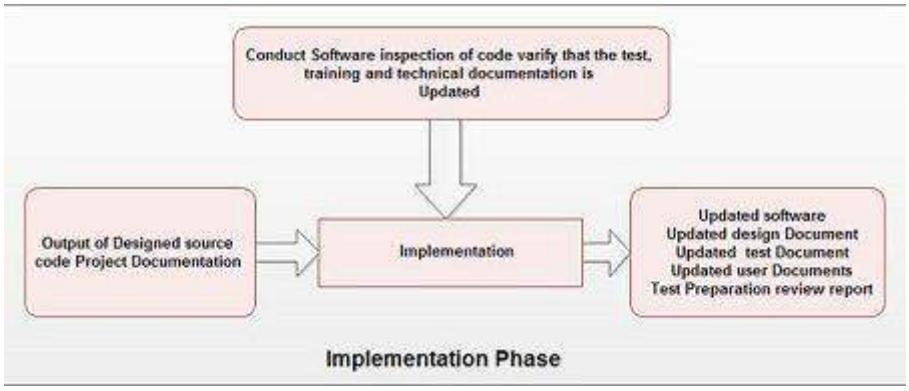

Figure 6. *Implementation.*

2.5. System Test Phase

In this phase, the regression testing (a type of system testing) is performed on the modified system to ensure that no new faults are introduced in the software as a result of the maintenance activity. The input attribute comprises the updated software documentation, test preparation review report, and the updated system.

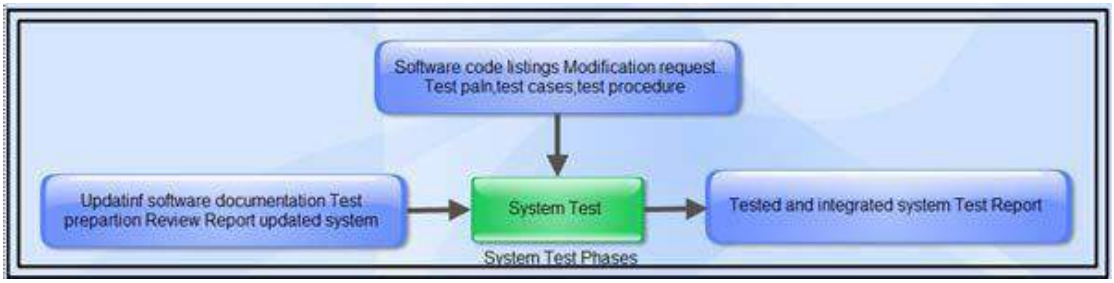

Figure 7. *System Test.*

2.6. Acceptance Test Phase

In this phase, acceptance testing is performed on the fully integrated system by the user or by a third party specified by the user. The objective is to detect errors and verify that the software features are according to the requirements stated in the modification request. The input attribute comprises the fully integrated system, acceptance test plans, acceptance test cases, and acceptance test procedures.

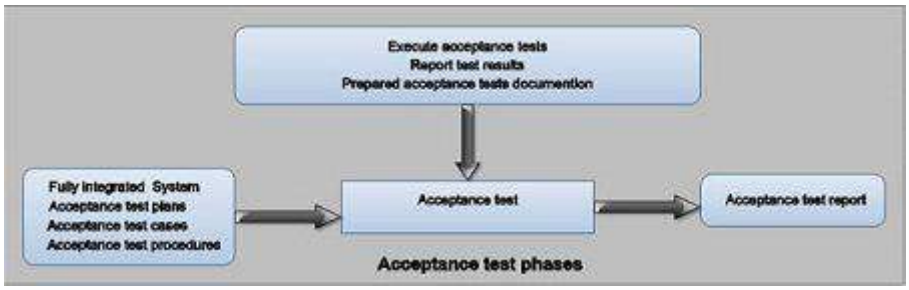

Figure 8. *Acceptance Test.*

2.7. Delivery Phase

In this phase, the modified (or new) software system is delivered to the user. In addition, users are provided with a proper documentation consisting of manuals and help files that describe the operation of the software along with its hardware specifications. The input attribute comprises a fully tested and accepted version of the system.

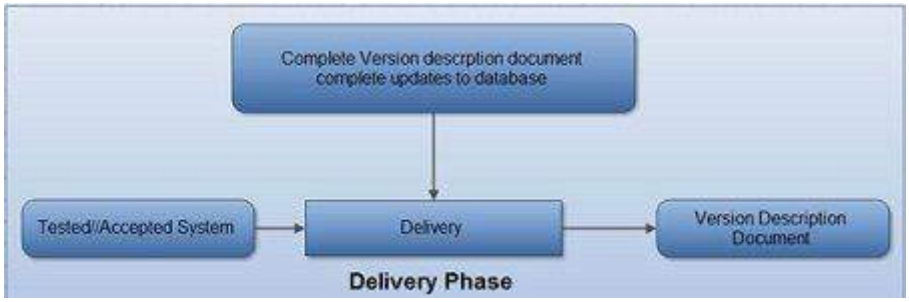

Figure 9. *Delivery.*

3. Causes of Software Maintenance Problems

3.1. 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.

3.2. Lack of Code Comments

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

3.3. 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.

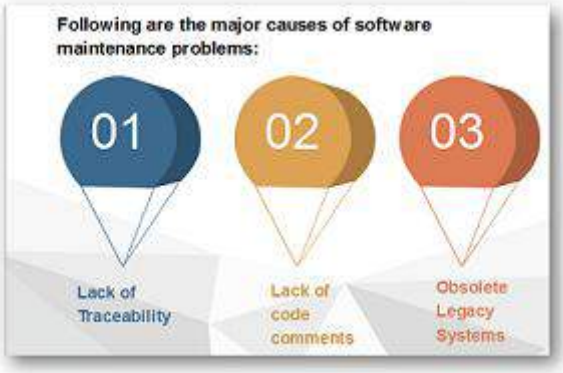

Figure 10. *Causes of Maintenance Problems.*

4. Why Software Maintenance?

Maintaining a system is equally important as Web Application Development. It keeps solutions healthy to deal with changing technical and business environment. Generally, IT service providers suggest their clients to go for software maintenance services for the consistent and enhanced performance of the system. Sometimes system maintenance involves improvements in the existing solution and at times there are requirements of new development as per the changing market needs.

Software maintenance is necessary for several reasons that are listed below:

4.1. Bug Fixing

The most important part of maintenance management is bug fixing. It is essential to run the software seamlessly. It should be done on a priority basis.

This process contains search out for errors in code and corrects them. The issues can occur in hardware, operating systems or any part of the software. This must be done without hurting the rest of the functionalities of existing software.

4.2. Capability Enhancement for Changing Environment

This maintenance part is done for the improvement in the current features and functions to make the system compatible for changing the environment.

It enhances software programs, work patterns, hardware upgrade, compilers and all other aspects that affect system workflow. Boost your system performances using a technically updated solution applying software maintenance services regularly.

4.3. Removal of Outdated Functions

The functionalities that are not in use anymore and unnecessarily occupying space in the solution actually hamper the efficiency of the system. Hence, the removal of outdated functionalities is necessary. Such elements of UI and coding gets removed and replaced with new features using the latest tools and technologies.

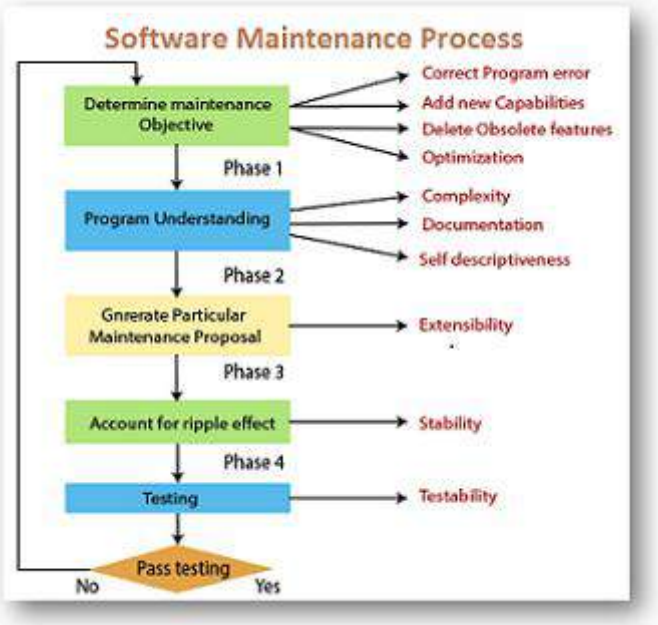

Figure 11. *Maintenance Process.*

This change makes the system adaptive to cope with changing circumstances.

4.4. Performance Improvement

Performance Improvement of a system is done to cope up with the new requirements. Data and coding restrictions, as well as re-engineering, are the part of software maintenance. It prevents the solution from vulnerabilities. This is not any functionality that performs in operations, but it develops to stop harmful activities like hacking.

5. Categories of Software Maintenance

Maintenance can be divided into the following:

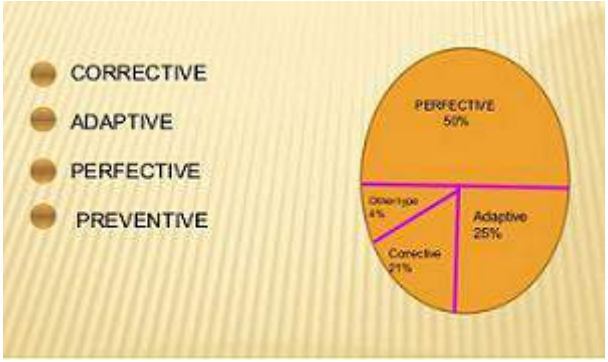

Figure 12. *Categories of Maintenance.*

5.1. Corrective Maintenance

Corrective maintenance of a software product may be essential either to rectify some bugs observed while the system is in use, or to enhance the performance of the system.

5.2. Adaptive Maintenance

This includes modifications and updations when the customers need the product to run on new platforms, on new operating systems, or when they need the product to interface with new hardware and software.

5.3. Perfective Maintenance

A software product needs maintenance to support the new features that the users want or to change different types of functionalities of the system according to the customer demands.

5.4. Preventive Maintenance

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

6. Software Maintenance Models

To overcome internal as well as external problems of the software, Software maintenance models are proposed. These models use different approaches and techniques to simplify the process of maintenance as well as to make is cost effective. Software maintenance models that are of most importance are:

6.1. Quick-Fix Model

This is an ad hoc approach used for maintaining the software system. The objective of this model is to identify the problem and then fix it as quickly as possible. The advantage is that it performs its work quickly and at a low cost. This model is an approach to modify the software code with little consideration for its impact on the overall structure of the software system.

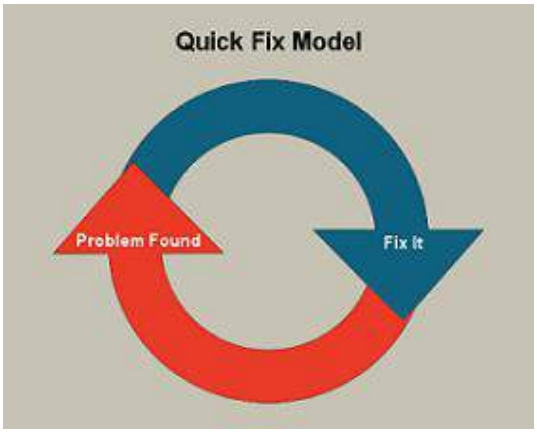

Figure 13. *Quick Fix Model.*

6.2. Iterative Enhancement Model

Iterative enhancement model considers the changes made to the system are iterative in nature. This model incorporates changes in the software based on the analysis of the existing system. It assumes complete documentation of the software is available in the beginning. Moreover, it attempts to control complexity and tries to maintain good design.

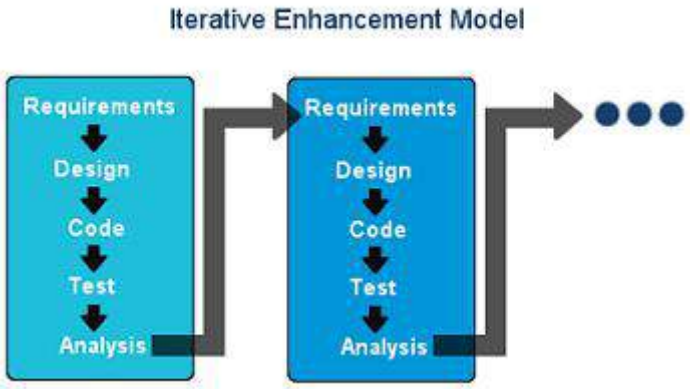

Figure 14. *Iterative Enhancement Model.*

Iterative Enhancement Model is divided into three stages:

a. Analysis of software system.

b. Classification of requested modifications.

c. Implementation of requested modifications.

6.3. The Re-use Oriented Model

The parts of the old/existing system that are appropriate for reuse are identified and understood, in Reuse Oriented Model. These parts are then go through modification and enhancement, which are done on the basis of the specified new requirements. The final step of this model is the integration of modified parts into the new system.

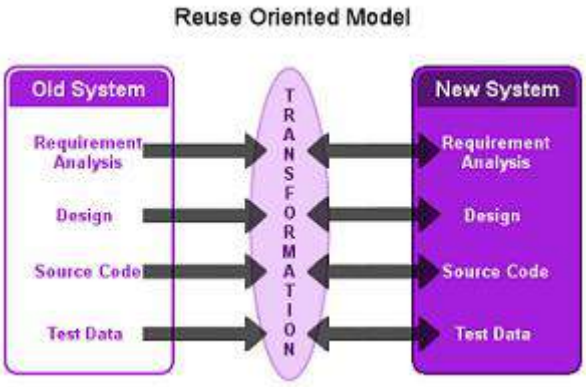

Figure 15. *Re-use Oriented Model.*

6.4. Boehm's Model

Boehm's Model performs maintenance process based on the economic models and principles. It represents the maintenance process in a closed loop cycle, wherein changes are suggested and approved first and then executed.

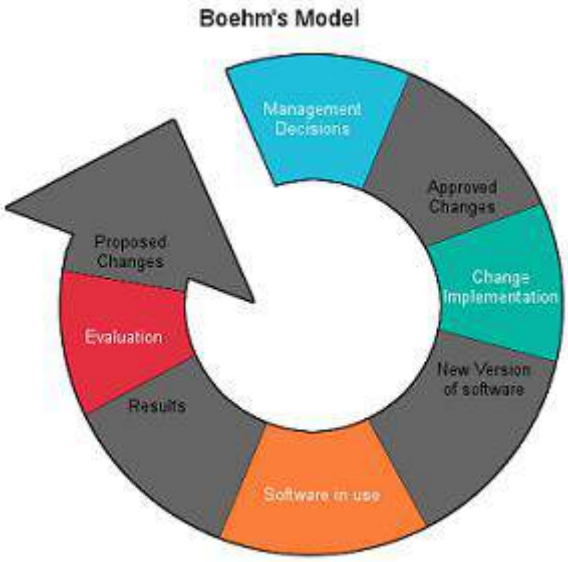

Figure 16. *Boehm's Model.*

6.5. Taute Maintenance Model

Named after the person who proposed the model, Taute's model is a typical maintenance model that consists of eight phases in cycle fashion. The process of maintenance begins by requesting the change and ends with its operation. The phases of Taute's Maintenance Model are: Change request Phase, Estimate Phase, Schedule Phase, Programming Phase, Test Phase, Documentation Phase, Release Phase, Operation Phase.

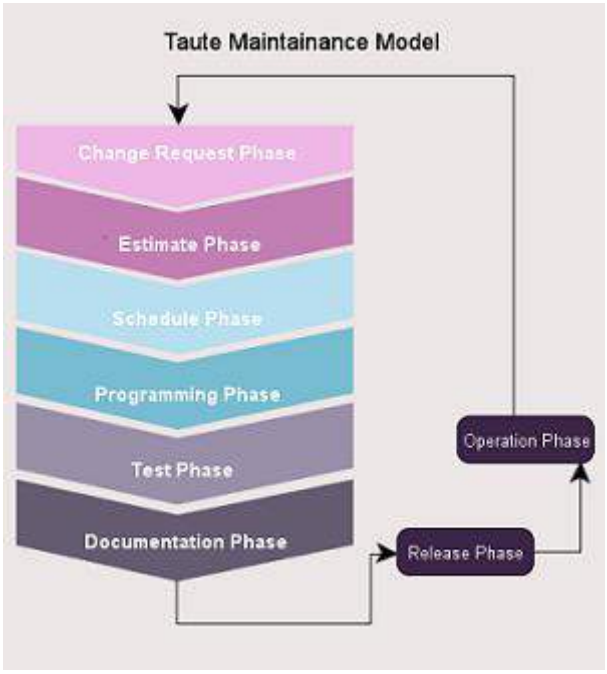

Figure 17. *Taute Maintenance Model.*

7. 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.

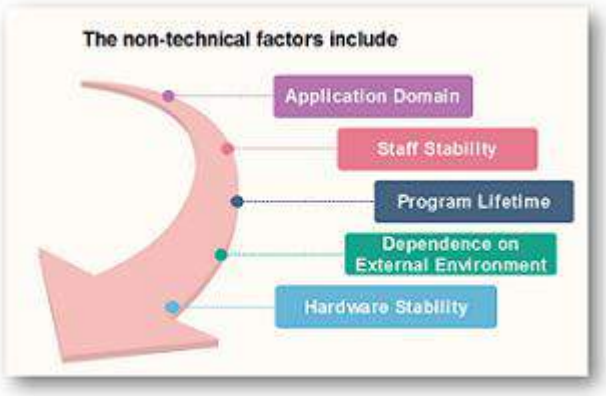

Figure 18. *Maintenance Cost Factors.*

7.1. 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.

7.2. 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.

7.3. Program Lifetime

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

7.4. 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.

7.5. 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.

7.6. Technical Factors

Technical Factors include the following:

Figure 19. *Technical Factors.*

7.6.1. Module Independence

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

7.6.2. Programming Language

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

7.6.3. Programming Style

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

7.6.4. 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.

7.6.5. 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.

7.6.6. 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.

8. Conclusion

It is not good to sign up for an annual maintenance without understanding the exact need. Thoroughly check the contract from all aspects before signing. Validate each and every point what is required for your business.

Software Maintenance is not an option, it is a must. Like take your car as an example, If you don't maintain it, it may cause many other issues every year. The amount for unperformed vehicle maintenance will cost you much more. Similarly, if you ignore the maintenance of a system, there will be a lesser scope for optimum business growth.

Through software maintenance, software systems can adapt to the changing technical environment and latest market trends. It also helps in predicting cash flow and controlling software expenditure. Hence, by adopting software maintenance developers can provide clients services that are up-to-date with the latest trends and is extremely beneficial.

Conflicts of Interest

The author declares that there is no conflict of interest regarding the publication of this article.

Funding

This research received no specific grant from any funding agency in the public, commercial or not-for-profit sectors.

References

[1] Pigoski, Thomas M. 1997: Practical software maintenance: Best practices for managing your software investment. Wiley Computer Pub. (New York).

[2] Kaur, U.; Singh, G. A Review on Software Maintenance Issues and How to Reduce MaintenanceEfforts. *International Journal of Computer Applications,* 2015, 118(1), 0975-8887.

[3] Bennett, K.H.; Rajlich, V.T., , May. Software maintenance and evolution: a roadmap. In *Proceedings of the Conference on the Future of Software Engineering.* 2000; pp. 73-87.

[4] Canfora, G.; Cimitile, A.; Lucarelli, P.B. Software maintenance. *Handbook of Software Engineering and Knowledge Engineering,* 2000, 1, 91-120.

[5] Debray, S.K.; Evans, W.; Muth, R.; De Sutter, B. Compiler techniques for code compaction. *ACM Transactions on Programming languages and Systems (TOPLAS),* 2000, 22(2), 378-415.

[6] Kidambi, P.C. Maintenance Issues in Software Engineering. Department of Computer Science Louisiana Tech University, 2003.

[7] Schneidewind, N.F. The State of Software Maintenance. IEEE Transactions on Software Engineering, pp. 303-310.

[8] Godfrey, M.W.; German, D.M. The past, present, and future of software evolution. Frontiers of Software Maintenance, 2008r; pp. 129-138.

[9] IEEE Std. 610.12. Standard Glossary of Software Engineering Terminology, IEEE Computer Society Press, Los Alamitos, CA.

[10] Pigoski, T.M. Practical Software Maintenance – Best Practices for Managing Your Software Investment. New York, NY, John Wiley & Sons.

[11] McDermid, J.A. Software Engineer's reference book: Oxford: Butterworth Heinemann, 1991.

[12] McClure, J.M.A.C. Software Maintenance: The Problem and Its Solutions: Englewood Cliffs, NJ: Prentice-Hall.

[13] Svahnberg, M. Supporting Software Architecture Evolution: Architecture Selection and Variability. Doctoral Dissertation, Department of Software Engineering and Computer, Science, Blekinge Institute of Technology, Karlskrona.

[14] IEEE. IEEE Standard for Software Maintenance. Software Engineering Standards Committee of the IEEE Computer Society, 1998.

[15] Chapin, N.; Hale, J.E.; Khan, K.M.D.; Ramil, J.F.; Tan, W.G. Types of software evolution and software maintenance. *J. Softw. Maint. Evol.: Res. Pract.* 2001, 13, 3-30.

[16] A. Bryant and J. A. Kirkham. *B. W. Boehm software engineering economics: a review essay. ACM SIGSOFT Software Engineering Notes*, 1983, 8, 44-60.

[17] Cimitile, G.C.A.A. Software Maintenance. Faculty of Engineering at Benevento, University of Sannio, Benevento, Italy.