Integrating Preventive Maintenance within the Product Life Cycle

Bindu Goel
University School of Information Technology,
GGS Indraprastha University,Dwarka Sector-16 C,
Delhi (INDIA)

ABSTRACT
The complexities associated with software system implemented often makes it imperative to maintain the existing software as a system of high functionality and throughput regime even in the face of changed external environment. It is preferable to enhance the functional operation by upgrading the existing software than changing entire systems as is demonstrated by continued usage of proprietary systems in large banking, insurance, logistic, shipping and airline systems etc. This makes preventive maintenance a very critical tool in the hands of organizations who wish to bring software reliability and maintainability in the future operations of the software systems. Preventive maintenance focuses on the re-engineering of the software rather than its operational maintenance. In this paper we attempt to analyze that preventive maintenance as an inevitable option for the maintenance of the software systems. There is also an attempt to integrate the preventive maintenance into the existing software product life cycle with the focus on the pre and the post deployment phase till the retirement of that software.

Keywords
Software maintenance, preventive maintenance, evolution, change impact, maintainability, documentation.

1. INTRODUCTION
With the maturity in the software industry, more and more software systems are entering a stage where the maintenance is crucial for the performance of the software. The process of changing a system after it has been delivered and is in use is called software maintenance [1]. The literal meaning of the word 'maintenance' for the hardware systems is restoring the system to its original state. However, software systems unlike other systems are ever evolving (changing due to various technological, architectural or external environmental factors). The software maintenance aims at bridging the gap between the existing operational system and the changing user specifications. Yet Lehman [2] established in his first law of software evolution that a software system that is used, undergoes continual change or becomes progressively less useful. Software maintenance and evolution are gaining importance because of ever increasing costs and ever decreasing speed of implementation. The activities are labor intensive and due to the repetitive and non creative nature they are not able to attract and retain best talent. We are entering into an era where increased resources will be required to maintain the existing deployed software systems than, what were utilized to develop them during the development phase. Software Maintenance is the most costly part of the software development life cycle with some authors placing these costs between 67% and 90% of total life cycle costs [3, 4].

Software maintenance is treated significantly different from software development. It is contested that software development continues to be requirement driven and is generally done without adequate regard to how changes will be incorporated post delivery of the software. Software maintenance in the present scenario is mostly event-driven and hence often performed as a 'reactive activity' as opposed to a 'proactive activity'. The absence of scheduling of the change interventions during software maintenance leads to higher cost and time. The development of new projects does not affect the organizations' daily operations while maintenance is intimately bound to disrupt its routine. The reason for this divergence is that during software development, the stages, meetings, follow-ups, etc. are previously scheduled and are backed with a high degree of motivation in the staff [5]. However, this situation does not exist in the maintenance, where the unrequited moments of edginess and pressures are multiplied, and the intensity of the staff dedication to these tasks goes beyond the desired level. This phase thus increasingly demands a more systematic and scheduled planning perspective. Software maintenance should be considered as an iteration of software development.

The paper discusses about the traditional methodologies. In the recent times object-oriented (OO) approaches provide a viable method to incrementally develop information systems, a host of new methods called agile development methodologies or lightweight methodologies claim to go a step further in overcoming the limitations of traditional plan-driven ones[6]. According to Boehm, "organizations must carefully evolve toward the best balance of agile and plan-driven methods that fits their situation [7]." Agile methodologies rely on speculation, or planning with the understanding that everything is uncertain, to guide the rapid development of flexible and adaptive systems of high value [8, 9]. Yet there are challenges in migrating to agile methodologies.

In this paper we present the current state of software maintenance with its reactive process implementation. We

attempt to identify the activities which could come under the preventive maintenance (PM) for software systems. We also attempt to define preventive maintenance accurately by analyzing its definition and how this can be integrated with Product life cycle (PLC). This integration can assist in decimating the maintenance cost and effort spent at one end while delivering significantly enhanced system functionality and uptime on the other.

The aim of the study described in this article is to investing ate the answers to the following questions:

- What is preventive maintenance?
- Whether it is necessary to do preventive maintenance?
- When does preventive maintenance start?

At the end of the paper, a model is proposed which integrates the preventive maintenance into PLC clearly identifying preventive maintenance activities to be performed during various phases.

2. PREVENTIVE MAINTENANCE

The latest IEEE standard [10] defines PM as the modification of a software product after delivery to detect and correct latent faults in the software product before they become operational faults. It does not address the operation of the software and its operational functions, e.g., backup, recovery, system administration, which are normally performed by those who operate the software. It is important to understand that PM is not a participant in the efficient operation of the software rather it is a preventive measure which works on the obsolescence of the software. PM is the planned maintenance of the software system that is designed to improve its reliability and maintainability, and reduce any unplanned maintenance activity. PM is performed by anticipating or forecasting problems which might occur in future. In other words preventive maintenance is done to modify software in order to improve the maintainability. Maintainability of software is the degree, to which it can be understood, corrected, adapted and/or enhanced [11]. PM includes

Designing a change to a software implemented system that can keep the system easy to maintain.

Continuously upgrading a system to enable it to cope with current and future changes:

by making the software less complex (Modularity),

and easier to interpret (Clarity)

Documenting the source code as well as appropriate supporting processes.

This is the process of changing software in order to improve its future operations or to provide a better basis for future enhancements. Recent technological advances in tools for inspection and reviews have enabled even more accurate and effective software maintenance. Miller (12) defines PM very aptly as the application of today's methodologies to yesterday's systems to support tomorrow's requirements.

PM is the planned and scheduled maintenance that also aims to postpone or reduce the failures of a system. Some researchers refer PM for software systems as a pro-active approach to operational software fault-tolerance and aims at counteracting the aging effect. [13]. Classically, software aging has two main symptoms: increased failure rate, and decreased service rate. Two types of software aging have been identified: Software product ageing and software process execution ageing [14]. Software product ageing is degradation in the software code and the documentation quality by continual usage. Software process execution ageing is the degradation in the performance characteristics of a software system through continuous running.

PM also concerns activities aimed at increasing the system's maintainability by updating documentation, adding comments, and improving the modular structure of the system [15]. PM work (essentially incremental re-engineering) can be supported to improve the system and make it easier to change. It has also been seen as a key element of enterprise risk management. Risks are assessed and identified so the necessary mitigating steps can be taken to prevent any unnecessary loss in the future.

2.1 Can software preventive maintenance be compared to hardware preventive maintenance?

PM till date has been forming one of the key activities for any hardware system maintenance while its role in software maintenance has not been fully appreciated, presuming that the software doesn't degrade with time [16]. Hence the performance of preventive activities for the hardware is a different concept. The hardware is made more reliable by using large number of components with the given reliability. These components are replaceable with components of better specifications in the system. While the confidence level i.e., the reliability in the software increases as the software evolves, instead of wearing out. So does the dependence on it. And the faulty modules are not easily replaceable due to various reasons discussed in [17]. The major reasons for the phasing out of the software can be attributed to its obsolescence. The various reasons to obsolescence can be technological, functional, and environmental such as [18]:

Change in infrastructure/technology

The software becomes very complex and extremely difficult to maintain as it evolves.

Deterioration in the code

Slow execution speed

Change in user requirements

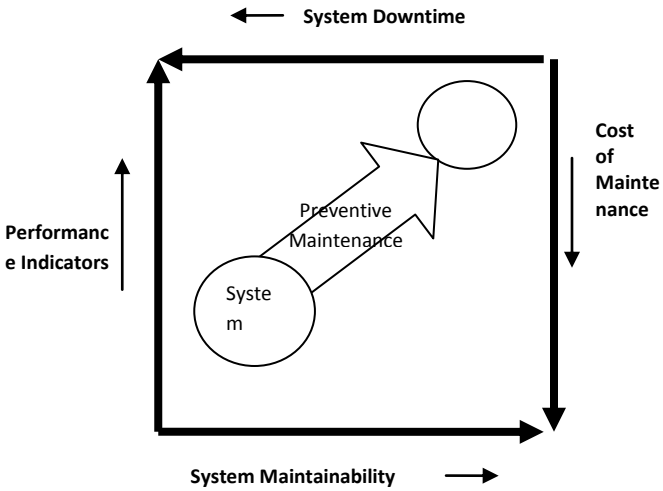
Poor graphical user interfaces

2.2 Whether it is even necessary to do preventive maintenance in software

The pressure during the development phase renders the preventive activity to a job of minimum importance as PM results in less time and incurs extra costs. However if one anticipates the limit of a software which requires change in future, it makes a lot of sense to prepare accordingly in advance. This perspective is usually missing. The maintenance is done

with the intention of making programs easier to understand and hence facilitates the future maintenance work [19].

Software developers always work under the target constraints in terms of time and budget. So they tend to behave extremely optimistic , thinking that everything will fall in place and go as per planning. While at the same time too much planning can actually kill the creativity of the developer. So anticipation in advance of certain things can help to plan the activities in the procedural manner. Most of the existing software can attain benefit from a good preventive maintenance program. It would be especially beneficial for those systems that rely on breakdown or run-to-failure maintenance.



Long-term benefits of preventive maintenance include as depicted in Figure 1.

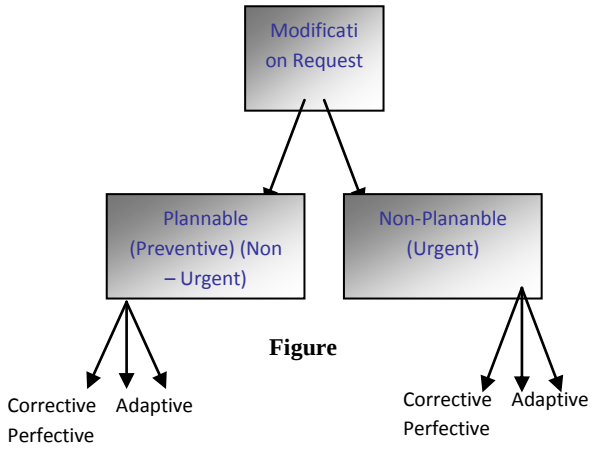
Figure 1. Benefits of PM

3. SOFTWARE MAINTENANCE CLASSIFICATION

Software maintenance consists of activities of correction i.e., to fix defects (corrective maintenance), adaptation i.e., to accommodate changes in external environment, e.g. OS, business rules (adaptive maintenance) and enhancement i.e., addition of other useful functions and extend software beyond original functional requirements (perfective maintenance). Preventive maintenance (PM) is defined as software reengineering needed because of deterioration due to change [7]. PM ensures that software are easily corrected, adapted andenhanced as per the requirement. Studies by Leintz and Swanson [20] show that 50% of the total maintenance efforts can be attributed to perfective maintenance, 25% for adaptive maintenance, whereas only 21% of the total efforts are attributed to corrective maintenance and 4% for preventive maintenance.

There has always been a debate on the classification of these types of maintenance [21]. This is partly due to the fact that many activities are difficult to classify. In addition, it is not unusual that while performing adaptive maintenance one finds a defect, or perhaps decides that some perfective rewriting is necessary to add a new feature [22]. Even when these categories are reasonably well defined, adaptive and perfective work often overlaps the corrective work. However, the two types, corrective and adaptive, along with perfective get the most attention.

Most maintenance are done using the quick-fix approach, because of the time pressure which hangs over long-term cost savings due to preserved technical quality [21]. Documentation is often not updated as the system is modified, and after a number of changes, the system's structure becomes complex that can not be further modified. The complex, unmanageable and deteriorating structure, increases unless reasonable efforts are made. This work is known as preventive change [19]. The software modification request is an understood fact which is inevitable as without this, the system will be rendered less useful. Here, it is interesting and important to classify the modification request and its causes. According to Pigoski [4], a maintenance process originates in a modification request submitted by a client, user or maintainer. These requests can be classified either as Repair or Enhancement. To understand the concept of PM under the umbrella of reactive software maintenance, we have classified the modification request as shown in figure 2.



Figure

2. Classification of Modification request

The modification request can be classified as either plannable (preventive) or non-plannable (corrective, adaptive, perfective). The urgent requests are generally user driven while the preventive requests are maintainer driven. The emergency or urgent modification requests cannot be planned and in such a case a workable solution rather than best solution is implemented. In these the documentation is generally not updated and code complexity increases which makes its maintenance difficult. The emergency requests can originate from following three reasons:

1. Urgent corrective action which requires fixing the bug to restore the system to normal operations. It is done to resolve the corrective, incidental issues and errors.
2. Urgent adaptive action done to the applications triggered by external changes. Changes in business rules, government policies and work patterns, are usually implemented on a deadline basis. So if a government regulation demands a change in a typical life insurance policy with a deadline of say 31st Jan 2008, then it will be immediately budgeted accordingly for a Jan release.
3. Urgent perfective action to enhance the functionality to better meet the business or user needs. This can be done to optimize and improve the performance indicators e.g., to speed up the existing slow software. It might be interesting to know that performance problems are treated very severely in live applications. Customers never like to see a slow system. It is always the vendor's endeavor to ensure speed ahead of perfection. This leads to compromise in the quality attributes of the software.

In non-emergency calls a planned and scheduled action can be implemented .It is done to plan and implement the action to avoid future problems based upon past incidents, anticipation, feedback and continuous product improvement. Software maintenance projects may be of widely varying size, they may be short in the correction of a localized error, or very long in the implementation of a new complex functionality. If transforming applications based on operating system changes is implemented, then it becomes a large and expensive project carried out over longer duration. Like a NT to XP migration project across applications. The following changes give good examples of the maintenance done in organizations from preventive perspective:

- Preventive Corrective Maintenance is performed for known serious defects. After deployment, a log of all defects is created. Defects are usually classified on the basis of severity. If a serious defect is discovered in the production stage of the application, then an over-riding code is usually put in. This is what is termed as a "Patch" in Microsoft terminology. "Patches" are examples of Corrective Preventive Maintenance when live software is protected against severe defects.
- Preventive Adaptive Maintenance is performed as restructuring and optimizing of software code and updating of the documentation. Software applications usually have infrastructure code in the background which acts as a backbone and enables interaction between applications. A change in infrastructure code is propagated and tested across all impacted applications. Thus, all applications are modified to prevent defects and errors coming from the code change. Infrastructure code prevents every possible code duplication and is highly re-usable, so the changes are thoroughly tested. Mostly, when an infrastructure code change goes to production, the other applications connected are modified accordingly to reflect changes. Designs for a group of applications should be modular enough to ensure ease of maintenance. So, it's the adaptive maintenance that is performed from a preventive point of view. Another example could be providing a newer front end with the same back end to facilitate the users.
- Preventive Perfective Maintenance is performed in special cases when the application is online from a B to C business model perspective where speed and responsiveness is the key. Another example could be that an organization maintains defect logs for existing known defects on an application catalog basis. As it can be inferred, live applications are far from perfect. In the case of perfective maintenance, the defects are sorted on basis of severity and cost to of correction. If software maintenance is being carried out, existing defects list is always referred to prevent any duplication of effort and repeating mistakes.

In the current practice, software maintenance is also driven by many other factors (managerial, customer etc.) which influence the maintainability of the software. The maintenance is driven by the planned budgets. Whenever a major change in applications is planned, an impact analysis is carried out across all concerned applications, which interact with the application in question. So when a major change is deployed, an account of the cost of maintaining all applications impacts the budget. All the maintenance is carried out on the release basis, which can be quarterly, once in 2 months or once in a month depending on the organization policies, business environment, budget size and team size.

Software maintenance is driven by the profile of maintainers. In some organizations project maintenance is performed by the development team or by the members of a separate organization or separate maintainers. The advantage of the development team is that they are versant with the project but are less careful in preparing the supporting documents. While the advantage of the separate maintenance team is that they give more attention to standards and high quality documentation. This helps the development team to concentrate on their ongoing projects. However the team should not be discouraged by giving extrapolating development assignments as rewards and giving more bonuses to development teams. The management attitude thus can play a major role in setting the right pitch for the maintenance job. The management should be able to assign priority to a request on the basis of its emergency and significance. Sufficient Training, tools and motivational environment should be provided to maintenance team. Also the time constraint should be realistic and achievable so that quality jobs are performed. Expertise of the maintenance team also plays a significant role. The technical expertise, functional knowledge of the business domain around the software under maintenance and the programmer attitude are the key attributes.

4. INTEGRATION OF PM INTO PLC

Does preventive maintenance start when the software is successfully deployed the client side? Or the anticipatory planning starts earlier. The focus in today's software development is to develop maintainable software. The criteria established apply to the planning of maintenance for software while under development, as well as the planning and execution

of software maintenance activities for existing software products. Software system analysis, development and implementation phases are critical for the success of the software. It seems feasible to integrate maintenance into the PLC as the PM. The reasons to this can be as

- The Corrective maintenance is error-driven, i.e., this activity is initiated by bugs and equivalent to debugging but it starts only when the system is operational.
- The adaptive maintenance is environment- driven. This activity is initiated when there are changes in the hardware, operating systems, files, compilers or external political or environmental changes. This also starts only when the system is operational.
- The perfective maintenance is primarily user driven. This is initiated in the form of enhancements and modifications done to meet the ever changing user requirements.

Although due to the definitions projected by IEEE std. [21, 22] there has been contention on how preventive and perfective maintenance seems overlapping and not clearly defined. Both are considered as the anticipatory activities. But the confusion can be contributed to the ambigious definition of preventive maintenance in the context of software systems. Discussion on this has been done by many researchers and practitioners in the field [23, 24]. The focus of both is to improve the maintainability of the system. It is suggested at this point that the preventive maintenance is done on scheduled basis while perfective maintenance can be user driven. The clear definition can help to classify the activities among various type of maintenance more clearly. It is only the preventive maintenance which begin before the system is operational. It is done to do the anticipatory planning and improve the future maintainability of the software.

Management of software quality while developing an innovative product is challenging, particularly in its early stages, because new technologies have several uncertainties in terms of the product characteristics and target clients. Therefore, it is difficult to set realistic goals for product quality, a requirement for quality management [24]. Additionally, developers tend to focus their attention on functionality in the early stages, while neglecting the other nonfunctional characteristics, such as performance, usability, reliability, installability, maintainability, and so on, in which customers are also interested. The work on nonfunctional characteristics is often relegated to the later stages of product development, after the initial excitement of a new technology is over. By this time, it may be too expensive to improve these characteristics, particularly if the systems involved are large and complex.

The Maintenance Plan should be developed in parallel with the Development Plan as depicted in Figure 3.. The maintainer should establish the required organizational interfaces in the plan. Preventive maintenance planning, and development scheduling are two activities that are inter-dependent but most often performed independently. Considering that preventive changes and repair affect both available up time, and the probability of system failure, it is surprising that this inter-dependency seems to be overlooked in the literature.

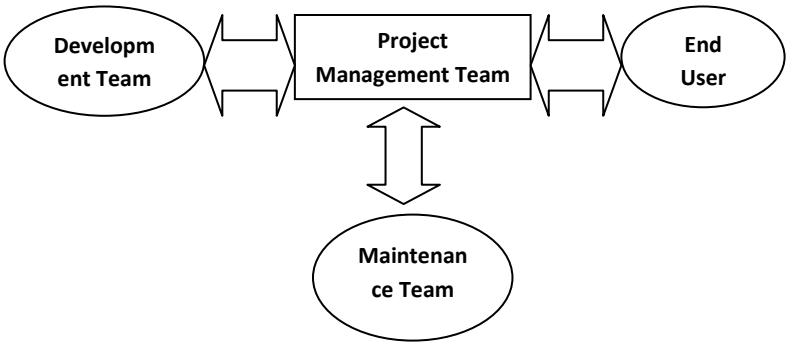


Figure 3. Maintenance team's role depiction

Figure 4. presents the proposed model for integration of PM into PLC. The model outline the preventive maintenance activities during each phase.

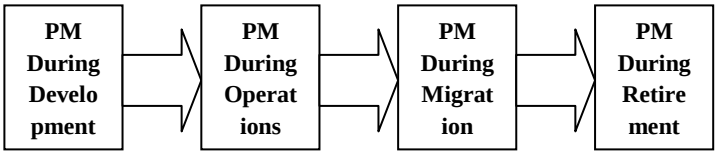
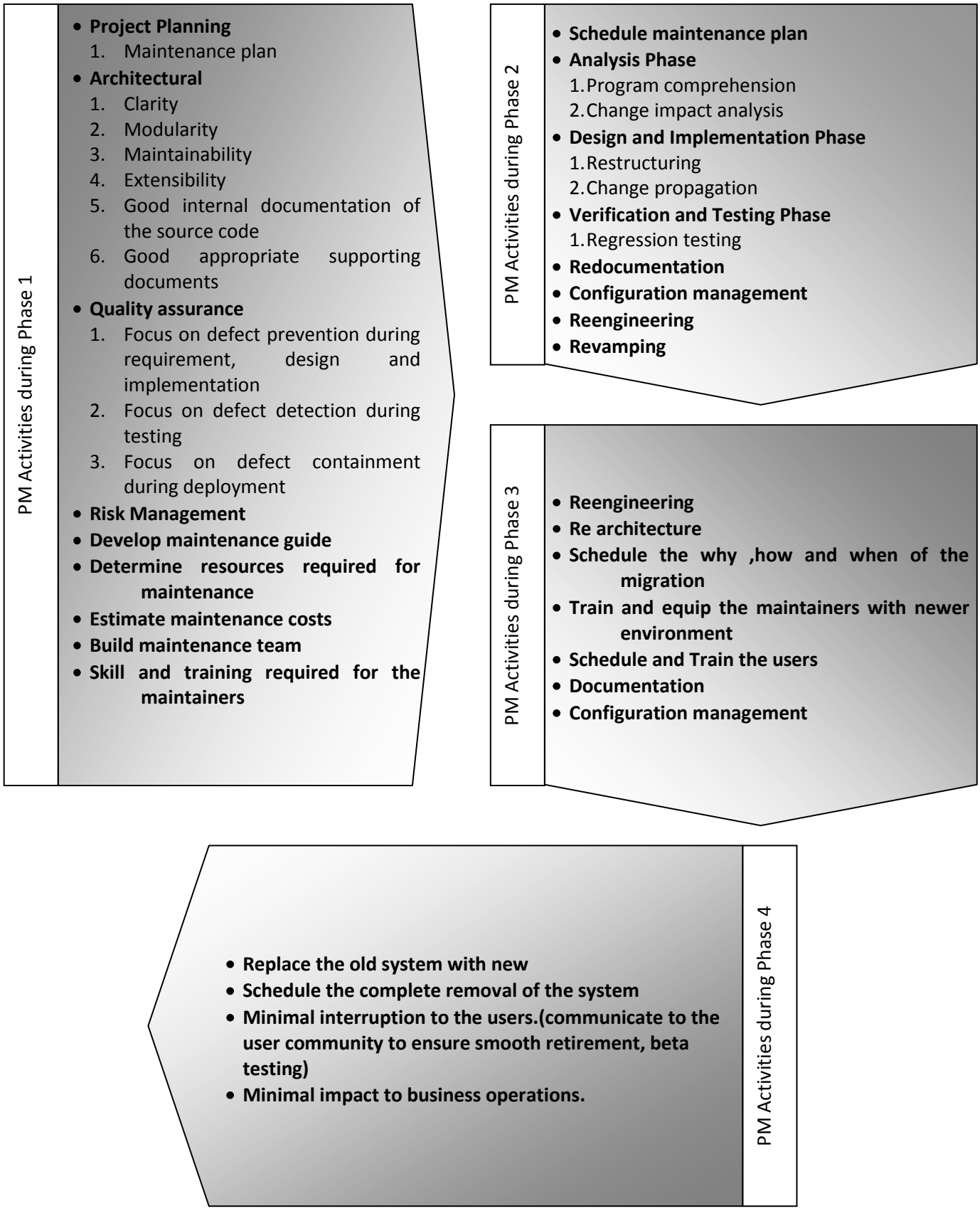


Figure 4. The Proposed model of integration of PM in PLC


PM Activities during Phase 1
• **Project Planning**
1. Maintenance plan
• **Architectural**
1. Clarity
2. Modularity
3. Maintainability
4. Extensibility
5. Good internal documentation of the source code
6. Good appropriate supporting documents
• **Quality assurance**
1. Focus on defect prevention during requirement, design and implementation
2. Focus on defect detection during testing
3. Focus on defect containment during deployment
• **Risk Management**
• **Develop maintenance guide**
• **Determine resources required for maintenance**
• **Estimate maintenance costs**
• **Build maintenance team**
• **Skill and training required for the maintainers**
PM Activities during Phase 2
• **Schedule maintenance plan**
• **Analysis Phase**
1. Program comprehension
2. Change impact analysis
• **Design and Implementation Phase**
1. Restructuring
2. Change propagation
• **Verification and Testing Phase**
1. Regression testing
• **Redocumentation**
• **Configuration management**
• **Reengineering**
• **Revamping**
PM Activities during Phase 3
• **Reengineering**
• **Re architecture**
• **Schedule the why ,how and when of the migration**
• **Train and equip the maintainers with newer environment**
• **Schedule and Train the users**
• **Documentation**
• **Configuration management**
• **Replace the old system with new**
• **Schedule the complete removal of the system**
• **Minimal interruption to the users.(communicate to the user community to ensure smooth retirement, beta testing)**
• **Minimal impact to business operations.**
PM Activities during Phase 4

5. PROPOSED MODEL FOR PREVENTIVE MAINTENANCE PROCESS

The Maintenance Process contains the activities and tasks necessary to modify an existing software product while preserving its integrity. These are the activities which are done to incorporate various modifications triggered by a set of change requests from system users, management or customers. The cost and impact of these changes are assessed to see how much of the system is affected by the change and how much it might cost to implement the change. The typical activities which comprise the scheduled maintenance process based on IEEE Std. 14764-2006 are as shown in the Figure 5. The activities are discussed briefly to understand the process of scheduled maintenance.

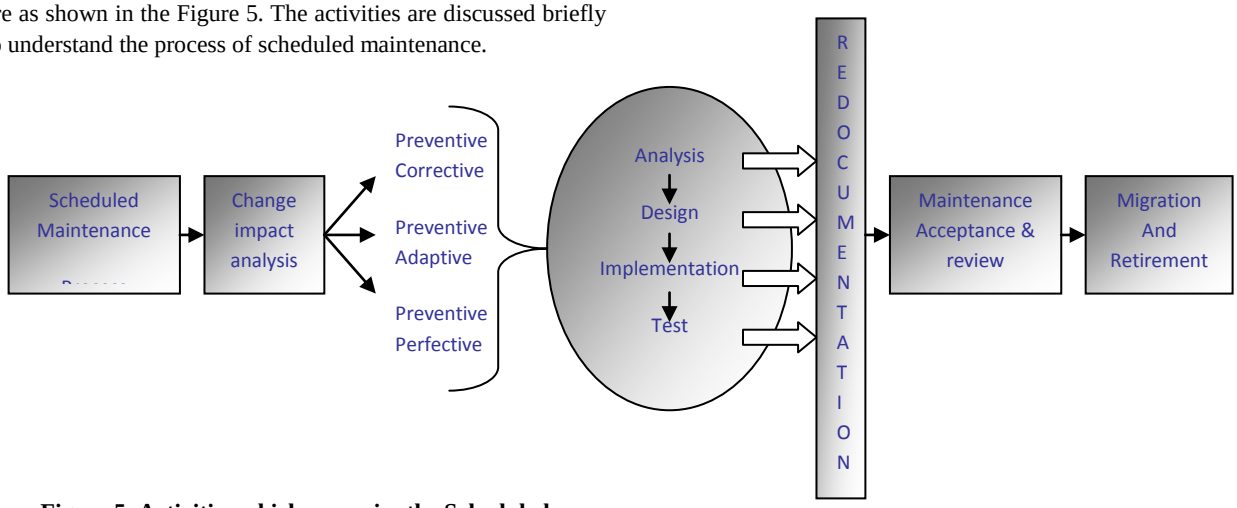


Figure 5. Activities which comprise the Scheduled Maintenance Process (PM)

a) Scheduled Maintenance Process. During this process implementation, the maintainer establishes the plans and procedures of various identified activities which are to be executed during the maintenance process. Maintenance plan is a document that identifies the management and technical approach to use in maintaining a software product on the scheduled basis. Typical topics include tools, resources, facilities, and schedules [25]. Setting of feasible target in terms of time, labor and motivation are planned.

b) Change impact Analysis. During this analysis, all system and software products that a change request impacts are identified. The impact of the change to all the related applications connected directly or indirectly is analyzed. This includes determining the scope of the changes to plan and implement work, accurately estimating the resources needed to perform the work, and analyzing the requested changes' cost and benefits.

c) Modification Implementation. This activity includes tasks to implement the modification request. The modification request can be classified under different types of maintenance as shown in the Fig 5. The modification can reinitiate in analysis phase, design and implementation phase and finally their verification and testing. The important task can be the re documentation of all the changes.

d) Maintenance Review& Acceptance. Include the activities to review the modification by the authorizing organization and getting the final consent. This includes the final approval after all changes and its impact among all the applications is thoroughly reviewed.

e) Migration. Include activities concerning the migration of the modified system to a new environment such as notifying when, why and how the migration will take place, train the users ,train the maintainers, analyze the impact of new system etc. During a system's life, it may have to be modified to run in different environments. In order to migrate a system to a new environment, the maintainer needs to determine the actions needed to accomplish the migration, and then develop and document the steps required to effect the migration.

f) Retirement: includes the tasks In order to retire a software product, the maintainer should determine the actions needed to accomplish the retirement, and then develop and document the steps required to effect the retirement. Consideration should be given to accessing data stored by the retired software product.

6. RECOMMENDED GUIDELINES FOR PREVENTIVE MAINTENANCE

We List below the guidelines that help prevent problems and assist the process of preventive maintenance at each step of the software lifecycle. They are classified into different heads.

6.1 Operational

- Debugging the software on scheduled basis by referring to the defect log created after the deployment.
- Maintainability i.e. one should periodically monitor system health and prevent system illness by checking the system maintainability level.
- Significant resources could be saved at the upfront maintenance process level through providing:
    a. Ongoing user training in relevant systems and their operations
    b. Written recovery restart instructions and notifications about known problems
    c. Software rejuvenation: the software is periodically stopped and restarted in order to refresh its internal state. This prevents or at least postpones the occurrences of failures. Through software rejuvenation implies overheads; it prevents more severe (and therefore more costly) failures.
- How long the various development tools will be viable.
- Updating and adding enhancements to the software on the scheduled basis.

6.2 Architectural

- Good programming style can reduce the impact of change and thereby reduce maintenance costs
- Language characteristics:
    - Language portability
    - Language legibility
    - Language stability
- Documentation of the source code and maintaining configuration files
- Rearchitecture: The transformation of a system by migrating it to a different technological architecture
    - Migration from old to modern programming language
    - Migration from old to new data bases
    - Modifying the user interface like providing GUI with back end remaining the same
- Restructuring: The transformation of system's internal structure without changing any external interfaces.

6.3 Managerial

- Motivating Preventive Maintenance Workers: A quality preventive maintenance program requires a highly motivated preventive maintenance crew.
- Establish inspection and preventive maintenance as a recognized, important part of the overall maintenance program.
- Assign competent, responsible people to the preventive maintenance program.
- Follow-up to assure quality performance and ensure to everyone that management does care.
- In addition to explaining the importance of a good preventive maintenance program and the benefits that can be derived from it, training is probably the most effective motivational tool available to the maintenance supervisor. Provide training in precision maintenance practices and training in the right techniques and procedures for preventive maintenance on specific equipment.
- Set high standards.

6.4 Others

- To facilitate preventive measures proper documentation at each level phase of development as well as maintenance phase is one of the strongest tools in the hands of the maintainer. Outsourcing companies facilitates documentation for existing software systems.
- The availability of software engineering and software test environments to assist in production, debugging, configuration management and the satisfaction of reliability and quality requirements;
- Writing reusable software has been proposed for promoting preventive maintenance e.g., availability of the Off the shelf reusable components
- Issuing corrected release announcements with feasible deadlines is also seen as form of preventive maintenance.
- Create a backup plan. If the data changes frequently, schedule backups that can occur frequently e.g., maintain a library of backups, based on information restoration needs. Ensure that data is stored correctly by periodically testing backups.
- Maintain a trend analysis to account for the predictable changes. For example, if the CPU utilization rate always increases by 50% during late morning, it can be assumed that the increase is normal for that server.
- Create a problem resolution notebook .If a problem occurs keep a log of the actions taken to resolve it. In the future, the information in the log can be helpful for any maintainer who can solve the same problem more quickly. This information also ensure the accuracy in any part of the replacement issue.
- Schedule PM for the time when the software is available i.e. there is no load on the system and thus typically results in lesser downtime and cost.

7. FINAL REMARKS

The objective of the paper is to understand the concept of Preventive or scheduled maintenance for the software systems. The current state of lower attention to PM can be attributed to various reasons like, not clearly defined state of practice, also the economic factors like shortage of time and high cost incurred leads it to activity of minimum importance. The fact remains in the old saying 'prevention is better than cure'.

In the current paper to understand the objective of PM better, we have revisited the classification of software maintenance and how preventive actions can be more organized and planned with more focus on the future maintenance of the system. Also a scheduled maintenance process is proposed which would substantially aid the creation of specialized tasks.

The PM on the other hand incurs an overhead in terms of downtime and this must be traded off with the downtime due to failures in obtaining the maximum benefits. This is an anticipatory job which is done by forecasting. But forecasting

the organization's competitive environment, user demands, regulatory changes, and external environment changes can be imprecise in some circumstances. A log or track of all changes should be referred before anticipating or implementing any change. This can help the managers to anticipate the maintenance needs better and recognize the consequences of prior enhancement or modification activities.

PM can be seen as correcting the code before the customer's complaint. It is seen as integrated with the average maintenance work. The release points can be referred as the check for the status of maintainability and provide the directions for the improvement. Therefore an objective understanding of maintainability and PM can provide a good basis for the maintenance of the software.

8. References

[1] Ian Sommerville, Software engineering (7th edition) Addison Wesley, 2004.

[2] Lehman, M. M. and Belady, L. Program Evolution: Processes of Software Change. London: Academic Press.1985.

[3] D.N. Card and R.L. Glass, Measuring Software Design Quality (Englewood Cliffs, 1990).

[4] T.M. Pigoski, Practical Software Maintenance. Best Practices for Managing Your Investment (Wiley,USA, 1997).

[5] Macario Polo, Mario Piattini and Francisco Ruiz, Integrating Outsourcing in the Maintenance Process Information Technology and Management Kluwer Academic Publishers. 3, 247–269, 2002.

[6] Sridhar Nerur, RadhaKanta Mahapatra, and George Mangalaraj, Challenges of Migrating to Agile Methodologies, COMMUNICATIONS OF THE ACM May 2005/Vol. 48, No. 5.

[7] Boehm, B. Get ready for agile methods, with care. Computer (Jan.2002), 64–69.

[8] Cockburn, A. Agile Software Development. Addison-Wesley, Boston,MA, 2002.

[9] Highsmith, J. Agile Software Development Ecosystems. Addison-Wesley, Boston, MA, 2002.

[10] IEEE Standard for Software Maintenance, IEEE Std 14764-2006. The Institute of Electrical and Electronics Engineers, Inc. 2006.

[11] Pressman RS. Software engineering (5th edition) McGraw-Hill Companies: New York NY, 2001.

[12] Miller, J.C., 1979," Techniques of Program and System Maintenance ", 1981, ed. Parikh, G., Winthrop Publishers, 181-182.

[13] Garg S, Puliafito A, Telek M, Trivedi K., Analysis of preventive maintenance in transactions based software systems. IEEE transactions on Computers, 47(1), pp. 96-107, 1998.

[14] Kajko-Mattsson M, Preventive Maintenance! Do We Know What It Is?, International panel, In Proceedings, International Conference on Software Maintenance, IEEE Computer Society Press in Los Alamitos CA, 2000.

[15] Van Vliet, H. 2000. Software Engineering: Principles and Practices, 2nd Edition. John Wiley & Sons, West Sussex, England.

[16] Kajko-Mattsson M, "Can we learn anything from Hardware preventive maintenance?". 2001 IEEE.

[17] Yogesh Singh and Bindu Goel, "A step towards preventive maintenance", ACM Sigsoft , Vol.32 no.4 ,July 2007.

[18] Aggarwal K.K. and Singh Y., "|Software engineering: Programs, documentation, operating procedures". New Age international publishers, 2005.

[19] Takang, A.A., AND Grubb, P.A. 1996. Software Maintenance Concepts and Practice. Thompson Computer Press London, UK

[20] Lientz BP, Swanson EB. Software Maintenance Management: a Study of the Maintenance of Computer Application Software in 487 Data Processing Organizations. Addison-Wesley Publishing Company: Reading MA, 1980.

[21] Ned Chapin," Software maintenance types-A Fresh view", IEEE 2000.

[22] Hatton, L.;How Accurately Do Engineers Predict Software Maintenance Tasks? Volume 40, Issue 2, Feb. 2007 Page(s): 64 – 69.

[23] IEEE Standard Glossary of Software Engineering Terminology, IEEE Std 610.12-1990 (1991 Corrected Edition). The Institute of Electrical and Electronics Engineers, Inc., 1994.

[24] Chapin N, "Do We Know What Preventive Maintenance Is?", In Proceedings, International Conference on Software Maintenance, IEEE, Computer Society Press in Los Alamitos CA, 2000.

[25] IEEE Standard for Software Maintenance, IEEE Std 1219-1998. The Institute of Electrical and Electronics Engineers, Inc. 1998.