ASIAN JOURNAL OF INFORMATION TECHNOLOGY

Journal Home Page: https://medwelljournals.com/journalhome.php?jid=1682-3915

Maintenance Approach to Reduce Efforts in Software Development Life Cycle

Mahtab Alam
Department of Computer Science, Noida International University, Noida, India

Key words: Software maintenance, scalability, reliability, mean time between failure (MTTBF), mean time to recover (MTTR)

Corresponding Author:
Mahtab Alam
Department of Computer Science, Noida International University, Noida, India

Page No.: 227-232
Volume: 18, Issue 10, 2019
ISSN: 1682-3915
Asian Journal of Information Technology
Copy Right: Medwell Publications

Abstract: Software development process is combination of several phases and maintenance is one of the major components of it. It has been considered that about 70% or more of the total software development budget cost is expending in maintenance of a software application. Software maintenance is a process of activities when an application after deployment to modify an existent software application due to some error or in anticipation of future problem. A number of works has been done to reduce the maintenance cost and effort of an application; herewith, we are proposing an integral approach to reduce the maintenance cost while designing the software application. In this particular research work, we are proposing different approaches at design phase to reduce the cost and effort of all kinds of software maintenance during entire life cycle of an application.

INTRODUCTION

Software maintenance is a set of activity performed when software undergoes modification to code and associate documentation due to a problem or the need for improvement (Pressman, 1997). By the laws of software evolution, maintenance decisions are aided by understanding what improves to system overtime. We are interested in change in size, complexity, resources and ease of maintenance. Software maintenance is become a major activity in the industry. A surveys and estimate made between 1988 and 1990 suggested that an average as much as 75% of a project software budget is devoted to maintenance activity over the life of the software. Software maintenance costs are the greatest cost incurred in developing and using a software system. Maintenance cost varies widely from application to application but an average they seem to be between 2.0-4.0 times developments costs for large software system (Lawrence, 2003).

Software maintenance is the degree to which it can be understood, corrected, adapted and/or enhanced. Software maintenance accounts for more effort than any other software engineering activity. When the changes in the software requirement are requested during software maintenance, the impact cost may be >10 times the impact cost derived from a change required during the software design, i.e., the cost to maintain one line of source code may be >10 times the cost of the initial development of that line (Somerville, 1997). Maintenance in the wildest sense of post development software support is likely to continue to represent a very large fraction of the total system cost (Lawrence, 2003). As more programs are developed the amount of effort and resources expanded on software maintenance is growing. Maintainability of software thus continues to remain a critical area in the software development era. Verification and Validation (V&V) for software maintenance is different from planning V&V for development efforts (Wallace and Daughtrey, 1988).

Maintenance may be defined by defining four activities that are undertaken after a program is released for use. First, activity is the corrective maintenance that corrects uncovered error after software is in use, adaptive maintenance, the second activity is applied when changes in the external environment precipitate modification to software. The third activity incorporate enhancement that are requested by customers and is defined by perfective maintenance where most of the maintenance cost and efforts are spent. The fourth and last activity is preventive maintenance which is in anticipation of any future problem. The maintenance effort distributions in Table 1 (Sunday, 1989).

Table 1: Anticipation of any future problem

Activity	Efforts (%)
Enhancement	51.3
Adaptive	23.6
Corrective	21.7
Others	3.4

Maintainability has been defined as effort of personnel hours, errors caused by maintenance actions, scope of effort of the maintenance action and program comprehensibility is subject to the programmer experience and performance (Sneed and Kaposi, 1990).

Cost factor is an important element for the success of a project. Cost in a project is due to the requirement of hardware, software and human resources. Cost estimates can be based on subjective opinion of some person or determined through the use of models (Jalote, 1997). Reliability-constrained cost minimization cost subject to a system reliability goal. Reliability of a system is presented as a function of component failure intensities as well as operation profile and component utilization parameters. Let n denote the number of software components. ρ denotes the system reliability target and $\tau>0$ be the mission time, the probability of failure free execution with respect to time interval $[0, \tau]$ to be at least ρ. We assume that $0<\rho<1$. The Total Cost (TC) of achieving failure intensities $\lambda 1, \lambda 2, ..., \lambda n$ and $R(\lambda 1, \lambda 2, ..., \lambda n, \tau) \geq \rho$ (Helander *et al*., 1998):

$$\lambda i \geq 0 \text{ for } I = 1, 2, ..., n$$

The purpose of software cost model is to produce the total development effort required to produce a given piece of software in terms of the number engineers and length of time it will take to develop the software. The general formula used to arrive at the nominal development effort by Carlo *et al*. (2002):

$$PM_{initial} = c.KLOC^{k}$$

Where:

PM	:	Person per month
KLOC	:	Thousand of line of code
C and k	:	Constant given the model

Software metrics are numerical data related to software development. Metric strongly supports software project management activities. They relate to the four function of management which are as follows:

Planning: Metric save as a basis of cost estimating, training planning and resource planning, scheduling and budgeting.

Organizing: Size and schedule metrics influence a project organization.

Improving: Metrics are used as a tool for process improvement efforts should be concentrated and measure the efforts of process improvement efforts.

Controlling: Metrics are used to status and track software development activities for compliance to plan.

The first step on the maintainability analysis using metrics is to identify the collection of metrics that reflects the characteristics of the viewpoint with respect to which the system is being analyzed and discard metrics that provide redundant information (Muthanna *et al*., 2000).

Object oriented technologies greatly influence software development and maintenance through faster development, cost saving and quality improvement and thus has become a major trend for methods of modern software development and system modeling (Chu *et al*., 2002). Class, object, method, message, instance variable and inheritance are the basic concept of the object oriented technology (Kan, 2003). Object oriented metrics are mainly measures of how these constructs are used in designed process. Classes and methods are the basic constructs for object oriented technology. The amount of function provided by object oriented software can be estimated based on the number of identified classes and metrics or its variables.

Improving the quantity and reducing the cost of products are fundamental objective of any engineering discipline. In the context of software as the productivity and quality are largely determined by the process to satisfy the engineering objectives of quality improvement and cost reduction, the software must be improved. Cost factor is the crucial aspects of project planning and managing. Cost overrun can cause customers to cancel the project and cost underestimate can force a project team to invest much of its time without financial compensation.

MATERIALS AND METHODS

Maintenance; A different opinion: The maintenance of software is affected by many factors such as the availability of skilled staff, the use of standardized programming languages and inadvertent carelessness in design. Implementation and testing has an obvious negative impact on the ability to maintain the resultant software. Additionally, some software organization may become maintenance bound, usable to undertake the implementation of new projects because all their resources are dedicated to the maintenance of old software. The opinion of programmers, managers and customers are as follows:

Programmer's opinion: According to programmer's opinion, a program with a high level of maintainability should consist of modules with loose coupling and high cohesiveness, simple, traceable, well structured, well documented, concurrent sufficiently commented code, well defined terminology of their variables. Furthermore, the implemented routines should be of a reasonable size, preferably <80 lines of code with limited fan-in and fan-out. Finally, the declaration and the implementation part of each routine must be strictly separated.

Program managers opinion: Program manager always aims at the limitation of effort spent during the maintenance process. They also focus on the high reusability of one program.

Customers opinion: Nowadays because of the high demand of the successful software systems and external changes, a high level of modification can be attributed to changes in requirement.

Design consideration; A better way to reduce cost and efforts: Several elements affect and shape the design of the application. Some of these elements might be non-negotiable and finite resources such as time, money and workforce. Other elements such as available technologies, knowledge and skills are dynamic and vary throughout the development life cycle. Analyze the high level design of a software system for the purpose of prediction with respect to change difficulty from the point of view of the testers and maintainers (Briand *et al*., 1993). The decision for scalability is set in the context of a software engineering environment (Han, 1997). Although, these elements influence the design of an application to some extent, the business problems dictates the capabilities application must have for a satisfactory solution such are as follows:

Design for scalability: Scalability is the capability to increase resources to produce an increase in the service capacity. A scalable application requires a balance between the software and hardware used to implement the application. The two most common approaches to scalability are:

Scaling up: Refers to achieving scalability by improving the existing servers processing hardware. Scaling up includes adapting more memory, more or faster processes or migrating the application to a powerful computer. Typically, an application can be scale up without changing the source code. In addition, the administrative efforts do not change drastically. However, the benefit of scaling up tapers off eventually until the actual maximum processing capabilities of the machine is reached as shown in Fig. 1.

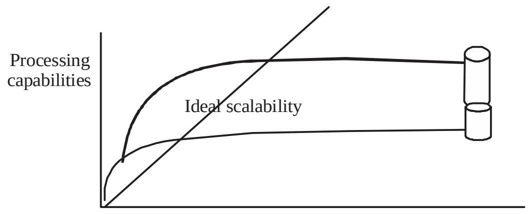

Fig. 1: Scaling up

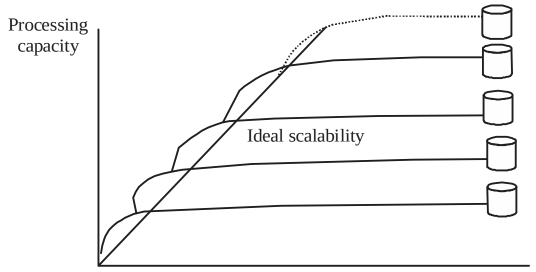

Fig. 2: Scaling out

Scaling out: Refers to distributing the process load across more than one server. This is achieved by using multiple computers; the collection of computers continues to act as the original device configuration from the end user perspective. The application should be able to execute without needing information about the server on which it is executing. This concept is called location transparency. It increases the fault tolerance of the application as shown in Fig. 2.

Design has more impact on the scalability of an application than the other three factors. As we move up the pyramid, the impact of various factors decreases as shown Fig. 3. To design for scalability, the following guidelines should be considered:

- Design process such that they do not waist
- Design process so that processes do not complete for resources
- Design processes for commutability
- Partition resources and activities
- Design component for interchangeability

Design for availability: Availability is a measure of how often the application is available to handle service requests as compared to the planned run time. Availability also takes into account repair time because an application that is being repaired is not available for use. The measurement types of availability is shown Table 2. The formula for calculating availability is:

$$\text{Availability} = (\text{MTBF}/(\text{MTBF}+\text{MTTR}))\times 100$$

The MTBF for a system that has periodic maintenance at a regular interval can be described by Mondro (2002) as follows:

Table 2: Measurement types for calculating availability

Names	Calculation	Definition
Mean Time Between Failure (MTBF)	Hours/Failure count	Average length of time the application runs before failing
Mean Time To Recovery (MTTR)	Repair Hours/Failure count	Average length of time needed to repair and restore service after a failure

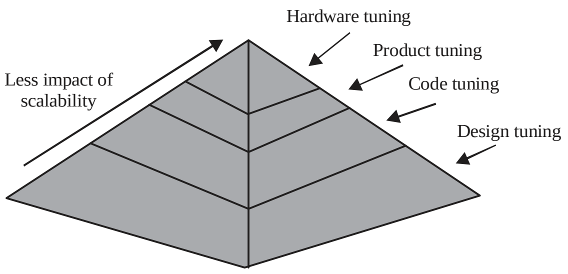

Fig. 3: Design pyramid

$$\text{MTBF} = \int_{0}^{T} R_T(t)dt / (1 - R_T(t)$$

where, $R_T(t)$ = exact reliable function assuming periodic maintenance every T (hours). Designing for availability include anticipating, detecting and resolving hardware and software failures before they result in service errors faults or data corruption thereby minimizing downtime. To design for availability of an application the following guidelines should be considered:

Reduce planned downtime: Use rolling upgrades, e.g., to update a component on a clustered server, we can move the server's resources group to another server, take the server offline, update the component and then bring the server online. Meanwhile, application experiences no downtime.

Use Redundant Array of Independent Disks (RAID): Raid uses multiple hard disks to store data in multiple places. If a disk fails, the application in transferred to a mirrored data image and the application continues running. The failed disk can be replaced without stopping the application.

Design for reliability: The reliability of an application refers to the ability of the application to provide accurate results. Although, software standard and software engineering processes guide the development of reliable or safe software, mathematically sound conclusions that quantify reliability from conformity to standard are hard to drive. Reliability measures how long the application can execute and produce expected results without failing. The following tasks can help to create reliable application.

- Using a good architectural infrastructure
- Including management information in the application
- Implementing error handling
- Using redundancy

Design for performance: Performance is defined by metrics such as transaction throughput and resource utilization. An application performance can be defined in terms of its response time. To define a good performance the following steps should be taken.

- Identify project constraints
- Determine services that the application will perform
- Specify the load on the application

Design for interoperability: Interoperability refers to the ability to operate an application independent of programming language, platform and device. The application need to design for interoperability because it reduces operational cost and complexity, uses existing investment and enables optimal deployment. To design application interoperability the following tasks should be considered:

- Network interoperability
- Data interoperability
- Application interoperability
- Management interoperability

Design for globalization: Globalization is the process of designing and developing an application that can operate in multiple cultures and locales. Globalization involves:

- Identifying the cultures and locales that must be supported
- Designing features that support those cultures and locales
- Writing code that executes property in all the supported cultures and locales

Globalization enables to create application that can accept, display and output information in different languages scripts that are appropriate for various geographical areas. To design for globalization the following information should be kept in mind:

- Character classification
- Date and Time formatting
- Number, currency, weight and measure convention

Design for clarity and simplicity: Clarity and simplicity are enhanced by modularity and module independence by structured code and by top-down design and implemented among other techniques. The allocation of functional requirements to elements of code represents an important step in the design process that critically impact modifiability (Munson, 1978).

The following guidelines for the definition of modules will have an extremely positive impact on maintainability:

- Use hierarchical module control structure whenever possible
- Each module should do its own housekeeping as first act
- Module should have only one entrance and exit
- Limit module size. Up to 200 statements
- Reduce communication complexity by passing parameters directly between modules
- Use 'go-to-less' or structured programming logic

Design for readability: Maintenance will ultimately result in changing the source code, understanding of thousand lines source code is almost impossible if source code is not well supported by meaningful comments. So, readability of the source code can be estimated by finding percentage of comments lines in total code. A new factor Common Ration (CR) is defined as Aggarwal *et al*. (2002):

- CR = LOC/LOM
- LOC = Total lines of code
- LOM = Total lines of commented in the source code

RESULTS AND DISCUSSION

The above practices of designing software will reduce the cost and efforts of maintaining software during maintenance phase of software development life cycle. Design of scalability will help to scale up and scale in of software at any time while design of reliability will helps to develop a software which will provide the desired output. Alam (2010) presented a software secure requirement metrics using a checklist in which all the security parameters proposed by Jan Jurjen are considered. With the help of these available checklists Degree of Secure Requirement (DSR) metrics can calculated the security concern of any proposed requirement (Alam, 2010; Bokhari and Alam, 2014).

CONCLUSION

Software maintenance is a set of activities when software undergoes for improvement due to an existing error or in anticipation of future problem. The life span of software entirely depends on a good design. Software design provides a blue print in which the entire software built. A good design software will be less problematic, so, software industries requires a good design paradigm and approaches for reducing the cost and efforts of software maintenance. In this study, we have proposed a number of design approaches for the same. In future, we will develop some design tool for better way of designing a software application.

REFERENCES

Aggarwal, K., Y. Singh and J. Chhabra, 2002. An integrated measure of software maintainability. Proceedings of Annual Reliability and Maintainability Symposium, Jan. 28-31, Seattle WA., pp: 235-241.

Alam, M., 2010. Software security requirements checklist. Int. J. Software Eng., (IJSE.), 3: 53-62.

Bokhari, M.U. and M. Alam, 2014. Security requirement for software quality-a survey of engineering discipline. Int. J. ICT. Manage., 1: 125-133.

Briand, L.C., S. Morasca and V.R. Basili, 1993. Measuring and assessing maintainability at the end of high level design. Proceedings of the International Conference on Software Maintenance, September 27-30, 1993, IEEE, Montreal, Quebec, Canada, pp: 88-87.

Carlo, G., J. Mehdi and D. Mandrioli, 2002. Fundamentals of Software Engineering. 2nd Edn., Prentice Hall, Upper Saddle River, New Jersey, USA.,.

Chu, W.C., C.W. Lu, C.H. Chang, Y.C. Chung, Y.M. Huang and B. Xu, 2002. Software maintainability improvement: Integrating standards and models. Proceedings of the 26th Annual International Conference Computer Software and Applications, August 26-29, 2002, IEEE, Oxford, UK., pp: 697-702.

Han, J., 1997. Designing for increased software maintainability. Proceedings of the International Conference on Software Maintenance, October 1-3, 1997, IEEE, Bari, Italy, pp: 278-286.

Helander, M.E., M. Zhao and N. Ohlsson, 1998. Planning models for software reliability and cost. IEEE. Trans. Software Eng., 24: 420-434.

Jalote, P., 1997. An Integrated Approach to Software Engineering. 2nd Edn., Springer, Berlin, Heidelberg, New York, ISBN: 9780387948997, Pages: 497.

Kan, K.S., 2003. Metrics and Models in Software Quality Engineering. 2nd Edn., Addison-Wesley, Boston, Massachusetts, USA., ISBN: 9780201729153, Pages: 528.

Lawrence, S., 2003. Software Engineering Theory and Practices. 2nd Edn., Pearson Education, Upper Saddle River, New Jersey, USA.,.

Mondro, M.J., 2002. Approximation of mean time between failure when a system has periodic maintenance. IEEE. Trans. Reliab., 51: 166-167.

Munson, J.B., 1978. Software maintainability: A practical concern for life-cycle costs. Proceedings of the International IEEE 2nd International Computer Software and Applications Conference (COMPSAC'78), November 13-16, 1978, IEEE, Chicago, Illinois, USA., pp: 54-59.

Muthanna, S., K. Kontogiannis, K. Ponnambalam and B. Stacey, 2000. A maintainability model for industrial software systems using design level metrics. Proceedings Of the 7th International Working Conference on Reverse Engineering, November 23-25, 2000, IEEE, Brisbane, Queensland, Australia, pp: 248-256.

Pressman, R., 1997. Software Engineering: A Practical Approach. 4th Edn., McGraw-Hill Education, New York, USA.,.

Sneed, H.M. and A. Kaposi, 1990. A study on the effect of reengineering upon software maintainability. Proceedings of the International Conference on Software Maintenance, November 26-29, 1990, IEEE, San Diego, California, USA., pp: 91-99.

Somerville, I., 1997. Software Engineering. 5th Edn., Addition Wesley, Boston.

Sunday, D.A., 1989. Software maintainability-a newility. Proceedings of the International Annual Reliability and Maintainability Symposium, January 24-26, 1989, IEEE, Atlanta, Georgia, USA, pp: 50-51.

Wallace, D.R. and T. Daughtrey, 1988. Verifying and validating for maintainability. Proceedings of the International Computer Standards Conference on Computer Standards Evolution: Impact and Imperatives, March 21-23, 1988, IEEE, Washington, DC., USA., pp: 41-46.