Software Maintenance Implications on Cost and Schedule

Bob Hunt, Bryn Turner, Karen McRitchie
Galorath Incorporated
100 North Sepulveda Boulevard
Suite 1801
El Segundo, California 90245
(703) 201-0651
bhunt@galorath.com

Abstract—Software Maintenance Implications on Cost and Schedule

The dictionary defines maintenance as, "The work of keeping something in proper order." However, this definition does not necessarily fit for software. Software maintenance is different from hardware maintenance because software doesn't physically wear out, but often gets less useful with age. Software is typically delivered with undiscovered flaws. Therefore, software maintenance is: "The process of modifying existing operational software while leaving its primary functions intact." Maintenance typically exceeds fifty percent of the systems' life cycle cost[1]. Additionally, software is highly dependent on defined maintenance rigor and operational life expectancy. Software maintenance generally includes sustaining engineering and new function development; corrective changes (fixing bugs); adapting to new requirements (OS upgrade, new processor); perfecting or improving existing functions (improve speed, performance); enhancing application with (minor) new functions (new feature.) Since software maintenance costs can be somewhat set by definition, the implications on cost and schedule must be evaluated. Development decisions, processes, and tools can impact maintenance costs. But, generally even a perfectly delivered system quickly needs upgrades. While software maintenance can be treated as a level of effort activity, there are consequences on quality, functionality, reliability, cost and schedule that can be mitigated through the use of parametric estimation techniques. [2] [3]

TABLE OF CONTENTS

1. INTRODUCTION.....1
2. SOFTWARE MAINTENANCE 1
3. APPROACHING THE MAINTENANCE ISSUE 2
4. SANITY CHECKS 3
5. FIVE ALTERNATIVE APPROACHES.....3
REFERENCES 6
BIOGRAPHY 6

1. INTRODUCTION

One of the greatest challenges facing software engineers is the management of change control. It has been estimated that the cost of change control can be between 40% and 70% of the life cycle costs[4]. Software engineers have hoped that new languages and new process would greatly reduce these numbers however, this has not been the case. This is fundamentally because software is still delivered with a significant number of defects. Capers Jones estimates that there are about 5 bugs per Function Point created during Development[5]. Watts Humphrey found "… even experienced software engineers normally inject 100 or more defects per KSLOC[6]. Capers Jones says, "A series of studies the defect density of software ranges from49.5 to 94.5 errors per thousand lines off code[7]." The purpose of this paper is to first review the fundamentals of software maintenance and to present alternative approaches to estimating software maintenance. A key element to note is that development and management decisions made during the development process can significantly affect the developmental cost and the resulting maintenance costs.

This paper addresses issues associated with software maintenance estimating and touches on development and management decision that can significantly affect maintenance costs.

2. SOFTWARE MAINTENANCE

Maintenance activities include all work carried out post-delivery and should be distinguished from block modifications which represent significant design and development effort and supersede a previously released software package. These maintenance activities can be quite diverse, and it helps to identify exactly what post-delivery activities are to be included in an estimate of

[1] Software Total Ownership Cost: Development Is Only Part of the Equation, Dan Galorath, Webinar presentation, 2007.

[2] 1-4244-1488-1/08/$25.00 ©2008 IEEE.

[3] IEEEAC paper#1098, Version 4, Updated 2007:10:26

[4] Practical Software Maintenance, Thomas Pigoski, page 30, Wiley Computer Publishing, 1997

[5] Geriatric Issues of Aging Software, Capers Jones, CrossTalk, Dec 2007, Vol. 20 No 12

[6] Humphrey, W.,"A Personal Commitment to Software Quality." Pittsburg, PA: The Software Engineering Institute (SEI)

[7] Jones, T.C. Programming Productivity. New York: McGraw-Hill, 1972

maintenance effort. Maintenance activities, once defined, may be evaluated in a quite different light than when called simply "maintenance". Software maintenance is different from hardware maintenance because software doesn't physically wear out, but software often gets less useful with age and it may be delivered with undiscovered flaws. In addition to the undiscovered flaws, it is common that some number of known defects pass from the development organization to the maintenance group. This bow wave of unclosed bugs is exacerbated when multiple versions of the same deliverable exist simultaneously. Accurate estimation of the effort required to maintain delivered software is aided by the decomposition of the overall effort into the various activities that make up the whole process.

In his book Software Engineering Economics, Barry Boehm defines maintenance as the process of modifying existing operational software while leaving its primary functions intact. The definition includes the following types of activity within the category of software maintenance:

• **Redesign and redevelopment** of smaller portions (less than 50% new code) of an existing software product.

• **Design and development** of smaller interfacing software packages which require some redesign (of less than 20%) of the existing software product.

• **Modification** of the software product's code, documentation, or data base structure.

The definition excludes the following types of activity from the category of software maintenance:

• Major redesign and redevelopment (more than 50% new code) of a new software product performing substantially the same functions.

• Design and development of a sizable (more than 20% of the source instructions comprising the existing product) interfacing software package which requires relatively little redesign of the existing product

• Data processing system operations, data entry and modification of values in the data base

IEEE and others generally identify four categories of Software Maintenance efforts.

a) Correct - maintenance is a change made in order to remove a fault

b) Adapt - maintenance is a change made in order to become suited to a different condition

c) Perfect - maintenance is a change made in order to improve

d) Enhance - maintenance is a change made to forestall or reverse a deterioration

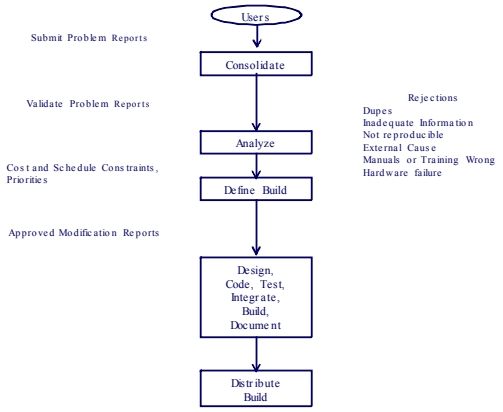
From the table below [8] it is obvious that maintenance related to enhancement or perfection of a software product is the largest single cost driver.

Repair Categories	**Percent of Total Maintenance Effort**
Enhance/Perfect	50%
Adapt	25%
Correct	20%
Prevent	5%

3. Approaching the Maintenance Issue

Maintenance is a complicated and structured process. In his textbook, Estimating Software Intensive Systems, Richard Stuzke outlines the typical software maintenance process. It is apparent that the process is more than just writing new code. The chart below summarizes the Stuzke process.[9]

The Software Maintenance Process*



* Estimating Software Intensive Systems; Richard Stuzke, 2005, Addison-Wesley, p 128

The following checklist can be used to explore the realism and accuracy of maintenance requirements.

• Which pieces of software will be maintained?

• How long will the system need to be maintained?

[8] Software Maintenance Concepts and Practices (second Edition) by Penny Grubb and Armstrong Takang, World Scientific, 2005, page 50

[9] Estimating Software Intensive Systems; Richard Stuzke, 2005, Addison-Wesley, p 128

• Are you estimating the entire maintenance problem, or just incremental maintenance?

• What level of maintenance is required?

• Is that which is being called maintenance in fact a new development project?

• Who will do the maintenance? Will it be done organically by the original developer? Will there be a separate team? Will there be a separate organization?

• Will maintainers be using the same tools used during development? Are any proprietary tools required for maintenance?

• How much Commercial-Off-The-Shelf (COTS) is there? How tightly coupled are the interfaces?

Some follow-on development may be disguised as maintenance. This will either inflate maintenance figures, or else cause shortfalls if basic maintenance gets pushed aside. These questions will help you ask whether maintenance is being honestly represented.

• Is the activity really an incremental improvement?

• Are healthy chunks of the original code being rewritten or changed?

• Will additional staff be brought in to perform the upgrade?

• Is the maintenance effort schedule regular and fairly flat, or does it contain staffing humps that look like new development?

4. SANITY CHECKS

Although sanity checks should be sought on a year-by-year basis, they should not be attempted for overall development. The reason for this is that maintenance activities can be carried on indefinitely, rendering any life-cycle rules useless. As an example, consider Grady (p. 17):

> We spend about 2 to 3 times as much effort maintaining and enhancing software as we spend creating new software.[10]

This and similar observations apply at an organizational level and higher, but not for a specific project. Any development group with a history will be embroiled in the long tail ends of their many delivered projects, still needing indefinite attention. Here are a few quick sanity checks:

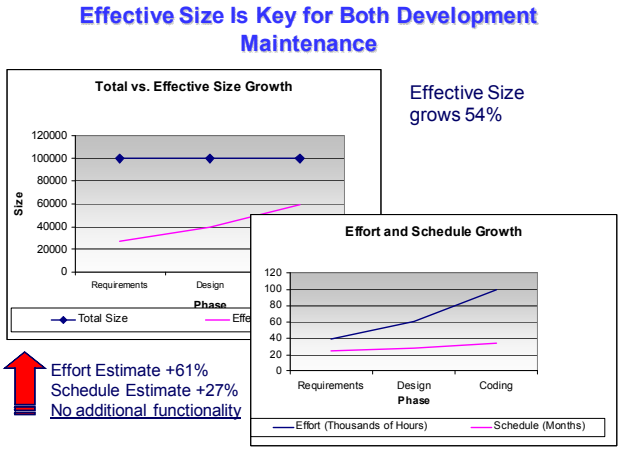
• One maintainer can handle about 10,000 lines per year.

• Overall life-cycle effort is typically 40% development and 60% maintenance.

• Maintenance costs on average are one-sixth of yearly development costs.

• Successful systems are usually maintained for 10 to 20 years.

Finally, as in development, the amount of code that is new versus modified makes a difference. The effective size, that is, the equivalent effort if all the work were new code, is still the key input for both development and maintenance cost estimation. This is demonstrated below in three scenarios, representing the effort impact between low, moderate and higher proportions of new code to reused code. (to use this last sentence you would need to replace the chart below with C3-50 from the May 07 era training material)

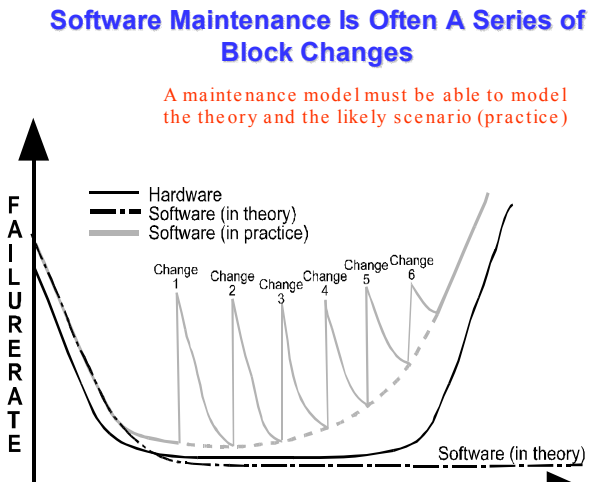


5. FIVE ALTERNATIVE APPROACHES

All software estimation techniques must be able to model the theory and the likely real world result. The real world scenario is presented in the figure below and points out that over time, the overlay of changes upon changes makes software increasingly difficult to maintain and thus less useful. Maintenance effort estimation techniques range from the simplistic level of effort method, through more thoughtful analysis and development practice modifications, to the use of parametric models in order to use historical data to project future needs.

3

[10] Grady, Robert P, Practical Software Metrics For Project Management and Process Improvement, Prentice Hall, Englewood Cliffs, NJ (1992)



5.1 Level of Effort

As is sometimes the case in the development environment, software maintenance can be modeled as a level of effort activity. As mentioned above, maintenance can be divided into several categories.

Lientz and Swanson[11] found the following typical proportions for the different types of maintenance:

Source	Application Type	Non-Comment Source Lines Per Year
Wolverton	Aerospace	4,000 to 8,000
Ferens-Harris	Aerospace	5,000 to 10,000
Daily	Real Time	5,000 to 30,000
Griffin	Real Time	6,000 to 12,000
Graver	Business (High Level Languages)	10,000 to 20,000
Rubin	Business	Up to 40,000
Overall Range		4,000 to 30,000

Given the repair category activities and the great variance shown in the table below, this approach clearly has deficiencies. In this approach, a level of effort to maintain software is based on size and type.

These are more precise figures for the number of lines per year that can be maintained by a single programmer, by application category.

Some caution must be applied when using these figures for an estimate:

- The definition of lines of code, function points, or other artifacts must be consistent
- Maintenance productivity (lines per programmer year) is often governed by the size of each Computer Software Configuration Item (CSCI) or work breakdown system item - larger CSCIs are more complex, and so more effort is required to maintain each line.
- Technical and personnel ability will influence maintenance.

5.2 Level of Effort Plus

Stuzke (see reference above) proposed that software maintenance starts with basic level of effort (minimum people needed to have a core competency and then that that basic core staff must be modified by assessing three additional factors; configuration management, quality assurance, and project management. His process (outlined on pages 135 to 141 of his textbook) addressed some of the additional factors affecting software maintenance.

5.3 Maintenance Change Factor

Software Cost Estimation with COCOMO II[12] (Boehm 2000) proposes a deceivingly simple, but also quite useful methodology for determining annual maintenance. Maintenance is one of the menu selections in the menu bar. In COCOMO II Maintenance encompasses the process of modifying existing operational software while leaving its primary functions intact. This process excludes:

- Major re-design and re-development (more than 50% new code) of a new software product performing substantially the same functions.
- Design and development of a sizeable (more than 20% of the source instructions comprising the existing product) interfacing software package

[11] B.P. Lientz and E.B. Swanson, Software Maintenance Management: A Study of the Maintenance of Computer Application Software in 487 Data Processing Organizations, Addison-Wesley, Reading, MA (1980)

[12] Barry Boehm, et al, Software Cost Estimation with COCOMO II, Prentice Hall, PTR, New Jersey (2000)

which requires relatively little redesigning of the existing product.
- Data processing system operations, data entry, and modification of values in the database.

The maintenance calculations are heavily based upon the Maintenance Change Factor (MCF) and the Maintenance Adjustment Factor (MAF). The MCF is similar to the Annual change Traffic in COCOMO81, except that maintenance periods other than a year can be used. The resulting maintenance effort estimation formula is the same as the COCOMO II Post Architecture development model:

$$PM_M = A \times (Size_M)^B \times \prod_{i=1}^{17} EM_i$$

As stated previously, three cost drivers for maintenance differ from development. Those cost drivers are software reliability, modern programming practices, and schedule. COCOMO II assumes that increased investment in software reliability and use of modern programming practices during software development has a strong positive effect upon the maintenance stage.
Annual Maintenance Effort = (Annual Change Traffic) * (Original Software Development Effort)

The quantity Original Software Development Effort refers to the total effort (person-months or other unit of measure) expended throughout development, even if a multi-year project.

The multiplier Annual Change Traffic is the proportion of the overall software to be modified during the year. This is relatively easy to obtain from engineering estimates. Developers often maintain change lists, or have a sense of proportional change to be required even before development is complete.

5.4 Managing Software Maintenance Costs by Developmental Techniques and Management Decisions During Development

When it comes to maintenance, "a penny spent is a pound saved." Better development practices (even if more expensive) can significantly reduce maintenance effort, and reduce overall life cycle cost. The more effort put into development, the less required in maintenance. As an example, the software development cost and schedule can be significantly impacted (reduced) by letting the number of defects delivered grow. This cost and schedule reduction is more than offset by the increase in maintenance cost. The following discussion is an example of how management decision can significantly affect/reduce software maintenance costs.

Lloyd Huff and George Novak of Lockheed Martin Aeronautics in their paper "Lockheed Martin Aeronautics Performance Based Software Sustainment for the F-35 Lightning II" propose a series of development and management decision designed to impact and reduce software maintenance costs. They propose an eight step process to estimate and control software maintenance[13]. Their proposed steps are:

1. Strive for Commonality

2. Apply Industrial Engineering Practices to Software

3. Engage

4. Adopt a Holistic Approach to Sustainment

5. Develop Highly Maintainable Systems and Software

6. Manage the Off-the-Shelf Software

7. Plan for the Unexpected

8. Analyze and Refine the Software Sustainment Business Case (use Parametric software sustainment cost estimates)

5.5 A Parametric Assessment of Software Maintenance

Parametric models like SEER for Software allow maintenance to be modeled in either of two ways:

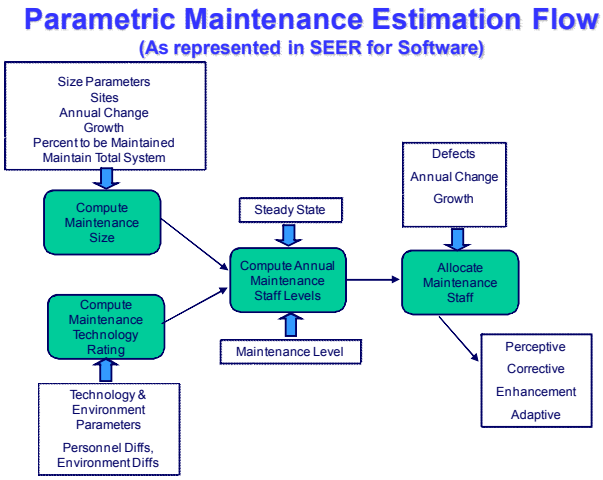
Estimating maintenance as a part of the total lifecycle cost**.** Choosing the appropriate Maintenance category parameters will include an estimate of maintenance effort with the development estimate for the individual software program. Several reports and charts show breakdowns of development vs. maintenance effort. This method is best used to evaluate life cycle costs for each individual software program.

Estimating maintenance as a separate activity**.** Using the appropriate maintenance parameters for the software to be maintained you can model the maintenance effort as a separate activity. This method will allow you to fine tune your maintenance estimate by adjusting parameters. Maintenance size should be the same as development size, but should be entered as all pre-existing code. This method can also be useful in breaking out total project maintenance costs from project development costs.

5

[13] Lloyd Huff, George Novak; Lockheed Martin Aeronautics; Lockheed Martin Aeronautics Performance Based Software Sustainment for the F-35 Lightning II

The general parametric approach is outlined in the flow diagram below.



As shown in the "Parametric Maintenance Estimation Flow" diagram, a good parametric estimate for maintenance includes a wide range of information. Critical information for completing a software maintenance estimate is the size or amount of software that will be maintained, the quality of that software, the quality and availability of the documentation, and the type or amount of maintenance that will be done. Many organizations don't actually estimate maintenance costs, they simply have a budget for software maintenance. In this case, a parametric model should be used to compute how much maintenance can actually be performed with the given budget.

Estimating and planning for maintenance are critical activities if the software is required to function properly throughout its expected life. Even with a limited budget, a plan can be made to use the resources available in the most efficient, productive manner. Looking at the diagram above, you can see that not only are the multiple inputs that impact the maintenance, but there are several key outputs that provide the information necessary to plan a successful maintenance effort.

6. Conclusion

The conclusions of this paper are:

- Software maintenance can be modeled using a simplistic method like Level of Effort Staffing, but this technique has significant drawbacks in that it does not account for changes in language or developmental processes.
- Software maintenance costs can be significantly affected by management decision during the developmental process.
- Software maintenance can be accurately estimated using parametric processes.
- Software maintenance is best modeled when development and management decisions are coupled with parametric cost estimation techniques.

REFERENCES

[1] Software Maintenance Concepts and Practices (second Edition) by Penny Grubb and Armstrong Takang, World Scientific, 2005.

[2] Estimating Software Intensive Systems; Richard Stuzke, 2005, Addison-Wesley.

[3] Lloyd Huff, George Novak; Lockheed Martin Aeronautics; Lockheed Martin Aeronautics Performance Based Software Sustainment for the F-35 Lightning II.

[4] G. Edward Bryan, "CP-6: Quality and Productivity Measures in the 15-Year Life Cycle of an Operating System," Software Quality Journal 2, 129–144, June 1993.

BIOGRAPHY

Bob Hunt *is Vice President for Services of Galorath Incorporated. Mr. Hunt is responsible for the management and technical direction of the services staff and the quality of the services products. Mr. Hunt has provided software program assessments, SEI Checklist evaluations, software sizing analyses, and software cost estimating for commercial and federal clients including the Customs and Border Patrol, the Department of Defense, NASA, and various commercial clients. Mr. Hunt's resource management experience includes cost analysis, should cost analysis, and competition analysis experience. His experience extends to traditional and non-traditional hardware and software weapons systems for major and non-major weapon systems acquisitions*