

THE DIMENSIONS OF MAINTENANCE

E. Burton Swanson
Graduate School of Management
University of California, Los Angeles
Los Angeles, California 90024

Keywords and Phrases

Software maintenance, performance measurement.

Abstract

The area of software maintenance has been described by one author as an "iceberg." (EDP Analyzer, 1972) Much goes on here that does not currently meet the eye. In part, this is the consequence of measurement difficulties. Practitioners and researchers can benefit from an understanding of the "dimensionality" of the maintenance problem. Some measures are suggested for coming to grips with this dimensionality, and problems of utilization associated with these measures are explored.

Introduction

The area of software maintenance has been described by one author as an "iceberg." (EDP Analyzer, 1972) By this term, we may infer that much goes on here that does not currently meet the eye, and further, that our ignorance in this regard is, in a sense, dangerous.

Boehm (1973) reports that according to one survey, almost 40% of the software effort in Great Britain now goes into maintenance. The "iceberg" is apparently big, and still growing.

The amount of time spent by an organization on software maintenance places a constraint on the effort that may be put into new system development. Further, where programming resources are cut back due to economic pressures, new development is likely to suffer all the more, since first priority must be given to keeping current systems "up and running."

So software maintenance is clearly a subject of importance. Yet we really know very little about it. It remains, indeed, an "iceberg."

In the context of a general concern about software "reliability," studies do exist which deal with the sources of errors in programs. (See, e.g., Boehm, et. al. (1975); Endres (1975); Miyamoto (1975); and Shooman and Bolisky (1975).) However, while these studies are often predicated upon a recognition of the problems of maintenance, their attention is for the most part on the development process. (See, e.g., Miyamoto (1975), on the "find-and-fix" cycle of bugs in software, during development testing.

Much of what has been reported on software maintenance consists of the individual experiences and

assessments of practitioners. (See, e.g., Ogden (1972); Lindhorst (1973); and Mooney (1975).) Sometimes these reports are based on purposeful empirical studies. (See, e.g., Brantley and Osajima (1975) and Stearns (1975).) Rarely, some ideas of theoretical import are ventured, in addition. (See especially Brooks (1975).)

For the most part, little research on software maintenance has been forthcoming. (For one notable exception, however, see Belady and Lehman (1975).)

In this paper, the problem of application software maintenance is directly examined. The intent is to come to grips with the "dimensionality" of the problem from both theoretical and practical points of view. The questions of measurement associated with the problem receive primary attention. Implications for managers and researchers are drawn.

Application software, rather than system software, has been chosen for study because, as is the case with the subject of maintenance, it has been relatively neglected.

The paper begins with a consideration of the bases for application software maintenance. An example of an organization structure for performing maintenance is then sketched, from which a maintenance data base is defined, and some measures of maintenance performance derived. The paper concludes with a consideration of the problems of utilization associated with measures such as those derived.

Bases of Software Maintenance

It is important to understand the bases of application software maintenance activity, i.e., the causes and choices which motivate it.

The causes for maintenance-type change have been variously described: e.g. "program won't run," "program runs but produces wrong output," "business environment changes," and "enhancements and optimization." (EDP Analyzer, 1972)

What is needed is a carefully constructed typology. Without making some important distinctions between types of maintenance activity undertaken, it will be impossible to discuss the effective allocation of these activities toward organizational ends.

The most basic cause of maintenance work is probably "failure" of the software. The most obvious type of failure is the processing failure, e.g., the abnormal termination of a program forcing job cancellation, or the production of "garbage" in an outputted report or file. Processing failures are those attributed to

"bugs" in the software. The failures may be precipitated by errors in input data, but the program itself is ultimately at fault, e.g., in neglecting to validate the input data. Processing fails to be completed, or, if completed, invalid outputs are produced.

Other forms of processing failures also exist, apart from those attributed to errors in the application software. Failure may be associated with the hardware or system software, for example. However, it is only failure in the application software which serves as a basis for application software maintenance.

A second type of application software failure may be termed the performance failure. Here the failure is in meeting performance criteria which have been specified in the system design. The software does not perform satisfactorily in terms of the functional specifications, and modification is called for to remedy the situation. Examples: an average inquiry response time exceeds some limit set, or an error rate in transaction processing is greater than that specified as permissible. No "bug" is necessarily involved. It may rather be a matter of loose coding, or the absence of "reasonableness checks" on computations performed. Further, the performance failure is likely to be a consequence, in part, of matters apart from the application software itself: the hardware and system software, operating procedures, and patterns of user behavior. Nevertheless, it is the application software which must be modified.

A third type of failure may also be identified. A program may be processed without error, and its performance to functional design specifications may be perfectly adequate. Nevertheless, there may exist certain failures in implementation. For example, programming standards may have been violated. Or inconsistencies or incompleteness in the detailed design, derived from the functional specifications, may be present. Often, such failures will lead in turn to processing and performance failures. But not always. Sometimes these failures will reflect sacrifices in software quality made precisely to achieve a performance level demanded, or a target date set. The implementation failure may thus remain hidden, unless audits are undertaken to insure the adherence of implementation practices to established organizational standards.

Maintenance performed in response to failures of the above types may be termed corrective maintenance. Especially where processing failures are concerned, a diagnosis of the causes of failure constitutes a significant portion of the task for this type of maintenance activity.

Corrective maintenance is an activity which would not be performed at all, were it not for the occurrence of failures. Thus, its costs must be compared with the opportunity costs of implementing more "failure-free" software.

Changes in the environment of a program typically lead to failures requiring corrective maintenance. However, such changes may also be anticipated, and the software adapted to their occurrence. Thus, failure may be avoided.

Two types of environmental change may be identified: change in data environment and change in processing environment.

Examples of change in the data environment would be a change in the classification code system associated with a particular data element, or the logical restructuring of a data base. These changes may motivate changes in data media employed or physical data

organization, but the basic change is in the data itself.

Examples of change in the processing environment would be the installation of a new generation of system hardware, necessitating recoding of existing assembler language programs; or, the installation of a new operating system version, necessitating modification of job control language statements employed in processing.

The earlier days of computer programming were marked by the extreme vulnerability of programs relative to changes in the data and processing environments. Subsequent developments in higher level languages, operating systems, and data base management systems have been directed in part toward insulation of programs from the effects of these changes.

Maintenance performed in response to changes in data and processing environments may be termed adaptive maintenance. The timely anticipation of environmental change is necessary to insure effective performance of this type of maintenance.

The amount of adaptive maintenance which must be performed on software is often a reflection of program "portability," i.e., the transferability of the program to new data and processing environments.

Failure and environmental change constitute "causes" of maintenance activity in the sense that a response is typically unavoidable if the program is to be kept operable. Other bases for maintenance exist, however, which are more a reflection of the initiatives of user and maintenance personnel.

Given that a program performs within functional design specifications (and thus no issue of performance failure exists), it may nonetheless be possible to improve the cost-effectiveness of this performance. Processing inefficiency may exist, e.g., in the use of an inferior computational algorithm, or inappropriate language features, or in making poor use of computer operator time.

Performance enhancement within established specifications may also be possible, e.g., in improving the readability of a report through reformatting, or in adding a new data element to those included in a report generated periodically.

Finally, although a program may be constructed and documented according to established standards (and thus no issue of implementation failure exists), it may nonetheless be possible to improve its general maintainability. For example, a program may be made more readable through insertion of certain comments, or it may be made similarly more accessible through a re-writing of its documentation.

An improvement in program maintainability is understood here to mean that the program will be more easily modified (in the course of corrective or adaptive maintenance) when it must be modified. It does not mean that program failures will occur less frequently, or that the effects of environmental change will be more easily avoided.

Maintenance performed to eliminate processing inefficiencies, enhance performance, or improve maintainability may be termed perfective maintenance. Its aim is to make the program a more perfect design implementation. It is undertaken when "justified," i.e., when the improvements to be achieved outweigh the costs of making those improvements.

In contrast to corrective and adaptive maintenance,

which serve merely to keep a program "up and running," perfective maintenance is directed toward keeping a program up and running at less expense, or up and running so as to better serve the needs of its users.

A summary of the bases of software maintenance is presented in Table 1.

TABLE 1

Summary

Bases of Software Maintenance

- A. Corrective
 - 1. Processing failure
 - 2. Performance failure
 - 3. Implementation failure
- B. Adaptive
 - 1. Change in data environment
 - 2. Change in processing environment
- C. Perfective
 - 1. Processing inefficiency
 - 2. Performance enhancement
 - 3. Maintainability

Organization for Maintenance

The measurement of any software maintenance activity will be found to be meaningful only within the context of an organizational structure for performing maintenance. One such hypothetical structure will now be described. It is highly simplified and intended to be illustrative only.

Let us assume that maintenance is organized separately from program development (and redevelopment) activity. A group of programmers exists which is collectively responsible for maintaining all production programs installed. The installation of a program follows a formal procedure, as does the installation of revised versions of a program based on redevelopment work.

A program is defined to be a separately compiled or assembled procedure. An application system typically consists of a family of programs.

Redevelopment work is based on revisions to the functional design specifications to which the program has been produced. Maintenance work is based on the functional design specifications associated with the installed version of the program.

All maintenance work is covered by "maintenance orders," which constitute an authorization to perform maintenance of various types on the programs installed. Each maintenance order is associated with a single "basis for maintenance" according to the classification scheme described earlier. An order covers maintenance on a single program, or on a group of programs.

"Open" maintenance orders exist on a continuing basis to cover all corrective maintenance required. Orders to cover adaptive and perfective maintenance are initiated as needed.

All changes to programs are formally made. A "change" consists of a program modification, involving the addition and deletion of source statements in a

single program, implemented as a unit. Each change is associated with a particular maintenance order. The "change level" of a program is incremented according to the changes made.

A change is made by a single programmer. The amount of the programmer's time spent on the change is formally recorded.

Changes in the documentation associated with a program are treated as changes to the program itself.

A Maintenance Data Base

It is now possible to imagine a maintenance data base which exists within an organizational structure such as that just described.

To facilitate the discussion of maintenance measurement to follow, one such data base is defined here, in third normal form. (Date, 1975) Again, the purpose is merely one of illustration.

The fundamental entities about which data is recorded are the program, the maintenance order, and the program change.

The domain for the relationships to be defined in the description of the three entities consists of the following:

| | |
|--------|---|
| PROG | program identification number |
| SOURCE | number of source statements in program |
| INSTR | number of machine language instructions in program |
| LANG | program language code |
| PIDATE | program installation date |
| RUNS | number of program runs undertaken since installation |
| FAILS | number of processing failures associated with program runs undertaken |
| LEVEL | program change level |
| CHANGE | program change identification number |
| ADD | number of source statements added by program change |
| DEL | number of source statements deleted by program change |
| PCHRS | number of person-hours spent in program change |
| PCDATE | program change date |
| PGMMR | programmer identification number |
| ORDER | maintenance order identification number |
| BASIS | maintenance basis code |
| OIDATE | maintenance order initiation date |
| OCDATE | maintenance order close date |
| CMHRS | cumulative number of person-hours spent in maintenance |
| NBEN | net benefits associated with maintenance performed. |

The relations defined on this domain are:

P: (PROG, SOURCE, INSTR, LANG, PIDATE, RUNS,

FAILS, LEVEL) with key: PROG

C: (CHANGE, PROG, ORDER, ADD, DEL, PCHRS, PCDATE, PGMMR) with key: CHANGE

O: (ORDER, BASIS, OIDATE, OCDATE, CMHRS, NBEN) with key: ORDER

This is a minimal data base only, and serves simply to indicate the nature of the foundation which must underlie the measures presented in the section to follow.

Note that the maintenance data base described corresponds directly to the organizational structure sketched in the previous section. Indeed, the data base is necessarily a reflection of this structure. Thus, for example, data gathered on program changes necessarily reflect the organizational procedure whereby changes are formally established and documented.

No argument is being made that the organizational structure and maintenance data base used here in illustration are uniquely appropriate for general real world application. Rather, it is that whatever choice is made, it is a single one, in the sense that the maintenance data which may be collected are more or less implied by the organizational structure established.

Measures of Maintenance Performance

In the context of a maintenance organizational structure and a particular set of data gathered in the performance of maintenance within this structure, it is possible to derive some performance measures which should be appropriately "suggestive" to management. ("Suggestive measurements," as defined by Churchman (1968), make only very weak assumptions about what a user wants. They make no pretense toward prediction, decision, or systemic evaluation for the user.)

Suppose that maintenance data of the type described in the previous section are gathered over some working interval, e.g., a month. A variety of summary data may now be generated (no order of importance is implied in the order of listing):

- S_1 : Number of programs maintained, as of end-of-period. (This is simply a count of the number of programs installed and covered by open orders for corrective maintenance.)
- S_2 : Total number of source statements maintained, as of end-of-period.
- S_3 : Total number of machine instructions maintained, as of end-of-period.
- S_4 : Average number of source statements per program maintained, in each programming language.
- S_5 : Average number of machine instructions per program maintained, in each programming language.
- S_6 : Percent of number of programs in each programming language.
- S_7 : Total number of program runs undertaken. (The sum of the run counts associated with the programs maintained, over the interval of measurement.)
- S_8 : Total number of processing failures occurring during program runs undertaken. (The sum of the failure counts associated with the programs maintained, over the interval of measurement.)
- S_9 : Average number of processing failures occurring per run undertaken. (This is computed as S_8/S_7 , and may be termed the "processing failure rate." An increase may be due either to external causes (e.g., the installation of new programs not sufficiently debugged) or internal effects (e.g., hasty modifications in maintenance, introducing new bugs).)
- S_{10} : Average age of programs maintained.
- S_{11} : Number of maintenance orders initiated, in each basis category.
- S_{12} : Number of maintenance orders closed, in each basis category.
- S_{13} : Number of maintenance orders open, in each basis category, as of end-of-period.
- S_{14} : Total net benefits associated with perfective maintenance completed. (Maintenance is said to be completed when the associated maintenance order is closed.)
- S_{15} : Total person-hours spent in perfective maintenance completed.
- S_{16} : Average net benefits associated per person-hour of perfective maintenance completed. (This is computed as S_{14}/S_{15} , and is one rough indicator of the "productivity of a person-hour of perfective maintenance.")
- S_{17} : Number of program changes made. (A simple count of the number of program changes made over the measurement interval.)
- S_{18} : Number of program changes made, in each maintenance basis category.
- S_{19} : Average number of program changes made per program maintained. (S_{17}/S_1)
- S_{20} : Total number of source statements added by program changes made.
- S_{21} : Total number of source statements deleted by program changes made.
- S_{22} : Net addition to total number of source statements maintained, due to program changes made. ($S_{20}-S_{21}$)
- S_{23} : Total number of person-hours spent in program change.
- S_{24} : Total number of person-hours spent in program change, in each maintenance basis category.
- S_{25} : Average number of person-hours spent per processing failure correction. (This is computed from those components of S_{18} and S_{25} which correspond to changes made due to processing failures. It is one indicator of the maintainability of the software, in the given maintenance environment.)
- S_{26} : Average number of person-hours spent per source statement added by program changes made. (This is computed as S_{20}/S_{23} , and is an alternative indicator of software maintainability, in the given maintenance environment.)

This list merely scratches the surface, of course. It represents only a crude attempt to derive some measures for coming to grips with the dimensionality of software maintenance. No special importance should be attached to the particular summary data listed. Only certain of these (perhaps S_9 , S_{16} , S_{25} , and S_{26})

approach the suggestive import one would look for in a good performance measure. Further, a good many other measures, some of them yet more interesting, may no doubt also be developed from the modest data base here defined. And, of course, the data base itself is easily extended, expanding further the opportunities for management-oriented measurement. However, the potential for performance measurement seems to me clearly established through this rather simple illustration.

But it should also be clear that a maintenance "iceberg" can never be definitively exposed through the generation of summary data such as those listed. Only a trace of a contour is suggested, through any one measurement. And it is not a contour bounded by the dimensions of space and time. The dimensions of maintenance are elusive indeed!

Further, it is not possible, *a priori*, to identify the performance measurements which will, in any situation, be the most useful for management purposes. Only in practice, within the context of ongoing organizational structures "in place," should it be possible to make such inferences with any degree of confidence.

Problems of Utilization

Several conclusions may be drawn from the discussion of the previous sections:

- (i) The measurement of maintenance performance presumes the establishment of a maintenance data base from which to derive the desired measures.
- (ii) A maintenance data base presumes the establishment of an organizational structure for performing maintenance, in terms of which the data of the data base are defined and collected.
- (iii) Measures of maintenance performance are not meaningful except within the context of the organizational structure(s) upon which they are based.

Consider for example, the summary measure S_{19} (average number of program changes made per program maintained) defined in the previous section. By definition, use of this measure presumes the existence of data on programs maintained and program changes made. However, the existence of such data is not sufficient to render the derived measures meaningful in a decision making context.

The interpretation to be attached to "programs maintained" and "program changes made" is ambiguous in the absence of familiarity with the organizational structure involved. For the definition of "program" constitutes an organizational choice, as does that of "program change." Both concepts are imbedded in the organizational procedures which require these fundamental entities to be recognized, identified, classified and described in data collection. Differing organizational procedures thus require differing interpretations of nominally-identical data.

The implications for the management of software maintenance are several. First, insofar as maintenance is performed informally, i.e. in the absence of an established set of organizational conventions and practices, the measurement of maintenance performance will not be feasible. The establishment of a maintenance data base, a precondition to performance measurement, is not possible in the absence of organizational structure. In informal situations, performance must be

assessed informally.

Secondly, in any given organization in which maintenance performance is formally assessed, in terms of specific measures, management must be thoroughly familiar with the organizational conventions and practices involved, in order to make intelligent decisions. No single measure or set of measures will itself reveal that maintenance is going "better" or "worse" than management has a right to expect or desire. Such a judgment must follow as a systemic inference from an analysis of the performance measurements, made in the context of familiarity with the organizational structure which made these measurements possible.

Finally, in the absence of familiarity with other, alternative organizational structures for performing maintenance, management will be essentially unable to assess its own established structure. Nothing in the performance measurements made within the context of the established structure will indicate the opportunity costs associated with this structure. Management must look outside, to the performance measurements and structures of other organizations, to assess what might reasonably be achieved through organizational change.

For researchers, the problems are similar. First, where maintenance is performed informally in organizations, and no maintenance data base thus exists, the gathering of data to support hypothesis testing will be greatly handicapped. The absence of "hard data" will necessitate rough estimates at best, and the conclusion drawn will have to be accordingly tentative.

One might limit research to those organizations possessing maintenance data bases, but the scope of inference would thereby be drastically narrowed. Nothing could be inferred about maintenance performed informally, and conclusions relative to cost-benefit trade-offs in formal and informal approaches to maintenance would not be possible.

Where maintenance data bases exist in organizations, researchers must be aware that these will not be directly comparable, for the most part. A standard organizational structure for performing maintenance does not exist. In each organization, a unique structure is likely to prevail. To employ maintenance data gathered from differing organizations, it will therefore be necessary to "translate" each into a common conceptual framework. (For example, what is understood to be the "number of programs maintained" in each organization must be translated into a well-conceived research definition.)

No "translation" of the type indicated will be possible without a study of the respective organizational structures actually in use. For the researcher, the gathering of maintenance data thus requires the additional gathering of data on organizational forms for performing maintenance.

Finally, only when the maintenance data of various organizations has been thus "standardized" according to the researcher's conceptual framework, will it be possible to truly assess the dimensions of maintenance. Only then will the shape and extent of the "iceberg" in each data processing organization be revealed. And only the may performance measures be established and employed to draw well-founded, research-based conclusions for general maintenance management.

References*

1. Belady, L.A. and Lehman, M.M.
"The Evaluation Dynamics of Large Programs."
IBM Thomas J. Watson Research Center.
Yorktown Heights, New York. September 9, 1975.
2. Boehm, B.W.
"The High Cost of Software."
Proceedings. Symposium on High Cost of Software.
Stanford Research Institute. September 1973.
3. Boehm, B.W., R.K. McClean and D.B. Urfrig.
"Some Experience with Automated Aids to the Design
of Large-scale Reliable Software."
Proceedings. International Conference on Reliable
Software.
21-23 April, 1975. Los Angeles, California.
4. Brantley, C.L. and Osajima, Y.R.
"Continuing Development of Centrally Developed
and Maintained Software Systems."
Proceedings. IEEE Computer Soc. Conf., Spring 1975.
5. Brooks, Frederick P., Jr.
The Mythical Man-month.
Addison-Wesley. 1975.
6. Churchman, C. West.
"Suggestive, Predictive, Decisive, and Systemic
Measurements."
Paper presented at the 2nd Symposium on Industrial
Safety Performance Measurement, National Safety
Council, Chicago. December, 1968.
7. Date, C.J.
An Introduction to Database Systems.
Addison-Wesley. 1975.
8. EDP Analyzer.
"That Maintenance 'Iceberg'."
Canning Publications. October, 1972.
9. Endres, Albert.
"An Analysis of Errors and Their Causes in System
Programs."
Proceedings. International Conference on Reliable
Software.
21-23 April, 1975. Los Angeles, California.
10. Lindhorst, W. Mike.
"Scheduled Maintenance of Applications Software."
Datamation. May, 1973.
11. Miyamoto, Isao.
"Software Reliability in Online Real Time Environ-
ment."
Proceedings. International Conference on Reliable
Software.
21-23 April, 1975. Los Angeles, California.
12. Mooney, John W.
"Organized Program Maintenance."
Datamation. February, 1975.
13. Ogdin, Jerry L.
"Designing Reliable Software."
Datamation. July, 1972.
14. Shooman, M.L. and M.I. Bolsky.
"Types, Distribution, and Test and Correction
Times."
Proceedings. International Conference on Reliable
Software.
21-23 April 1975. Los Angeles, California.
15. Slaughter, Dr. John B.
"Understanding the Software Problem." Report of
Workshop 1.
Proceedings. Symposium on the High Cost of Soft-
ware.
16. Stearns, Steven.
"Experience with Centralized Maintenance of a
Large Application System."
Proceedings. IEEE Computer Soc. Conf., Spring 1975.

*The author wishes to acknowledge the research assis-
tance of Mr. Gerry Tompkins in the identification of
items included among the references.