58.499 RESEARCH REPORT

SOFTWARE MAINTENANCE MANAGEMENT :

A SYSTEMATIC APPROACH

A research report presented in partial fulfilment of the requirements for the degree of Master of Business Studies (MBS) at Massey University

ANG HAK SENG

[1985]

TABLE OF CONTENTS

CHAPTER		PAGE
INTRODUCTION		1
1	SOFTWARE MAINTENANCE LIFECYCLE	4
2	SOFTWARE MAINTENANCE PROBLEMS	10
3	SOFTWARE MAINTENANCE CLASSIFICATION	16
4	FACTORS AFFECTING SOFTWARE MAINTENANCE	22
5	FRAMEWORK FOR SOFTWARE MAINTENANCE MANAGEMENT	30
	5.1 SOFTWARE MAINTENANCE OBJECTIVES	32
	5.2 SOFTWARE MAINTENANCE STRATEGIES	35
	5.3 SOFTWARE MAINTENANCE INFORMATION SYSTEMS	42
	5.4 SOFTWARE MAINTENANCE MEASURING METHODOLOGIES	54
	5.5 SOFTWARE MAINTENANCE METHODS	75
	5.6 SOFTWARE MAINTENANCE FUNCTION ORGANIZATION	159
6.	THE HUMAN ELEMENT IN SOFTWARE MAINTENANCE	167
7	MAINTAINING APPLICATION PACKAGES	174
8.	SUMMARY AND CONCLUSION	178
BIBLIOGRAPHY		182

INTRODUCTION

Ten years ago, the concept of software maintenance management was seldom discussed. This is no longer true today. In fact it is now rare to read any software engineering journal without some reference to the impact of software maintenance management.

There are three reasons that contribute to the increasing importance of this concept.

1. INCREASES IN SOFTWARE DEVELOPMENT AND MAINTENANCE COSTS

It is estimated that in 1973, the total software costs (both development and maintenance) in the U.S. alone was $20 billion. This figure is projected to increase 10 fold to 1985 ($200 billion)(1). Of these massive software costs, 40-95% are attributed to software maintenance. For example, a recent study by the U.S. Defence Department revealed that the development costs for Airforce avionics software averages $75 per instruction while the maintenance cost lies in the region of $4000 per instruction(2).

While the research done by various authors(3,4,5) show wide variance due to sampling error, different research methodologies and measurement problems, the economic importance the maintenance activities has been strongly indicated.

2. INCREASED COMPLEXITY OF APPLICATION SOFTWARE

The use of computers has increased dramatically in recent years. For example, there is a move from using computers for simple numeric calculations to complex financial modelling. This increase in software sophistication is also accompanied by an increase in software complexity and hence maintenance complexity. Together with increasing costs and complexity in software maintenance, there has been an upsurge in software application usage. Expectations of higher output and better performance have accentuated the need for better software maintenance management.

3. TECHNICAL PROGRESS AND OBSOLESENCE

In an environment of rapidly changing technology (both hardware and software), DP managment is faced with the dilemma of whether upgrading the existing systems to be in line with currenct state of arts and requirements or to completely rewriting the application system.

On one hand, rewriting an application system involves expensive capital outlay while, on the other hand maintaining current

software beyond useful life is wasteful. The uncertainty has resulted in a move toward formalizing the software maintenance function.

MOTIVATION FOR THIS RESEARCH

As high as 75% of computer resources is expended in the maintenance process and yet, very little is known about the nature of the maintenance problems and their solutions. This lack of knowledge is deplored by interested observers from the academic as well as the DP community. As one academic put it "software technologies of the 1960 and 1970's and those proposed for the 1980's mostly forcus on the technical aspects of software development, for the most part ignoring the end user, manangement issues and maintenance problems"(6).

The tremendous interest in software maintenance methodology by the DP community is demonstrated in a survey(7) of DP managers from both manufacturing and non manufacturing industries. About 70% of the participants in survey viewed the maintenance of existing systems equal or more important than new project development.

THE OBJECTIVE OF THIS STUDY

The objective of this research is to provide an exploratory study of the concept of software maintenance management. This study first examines the various classifications of software maintenance. This is followed by a discussion of the problems and factors facing the maintenance function. Next, a detailed framework necessary for effective software maintenance management is presented. Basically this framework can be categorized into the following 5 elements:

1. software maintenance lifecycle

2. software maintenance problems

3. software maintenance classification

4. factors affecting software maintenance

5. framework for software maintenance management

    5.1 software maintenance objectives
    5.2 software maintenance strategies
    5.3 software maintenance information systems
    5.4 software maintenance measurement methodologies
    5.5 software maintenance methods
    5.6 software maintenance function organization

Following which is a detailed discussion of human resources management in software maintenance and contractual issues in

relation to ready-make packages.

This study concludes by advocating a system approach to maintenance management and provides justification for such a choice.

REFERENCES

1. Boehm B.
"Software and Its Impact : A Quantitative Assessment"
Datamation 19, No. 5, May 1973 pg 48 - 59

2. DeRoze B. and Nyman T.
"The Software Life Cycle : A Management and Technological Challenges In The Department of Defence"
IEEE Transactions on Software Engineering SE-4,4 July 1978 pg 309-318

3. Riggs R.
"Computer System Maintenance"
Datamation November 1969 pg 227 - 235

4. Lientz B. P., Swanson E. B. and Tompkin G. E.
"Characteristics of Application Software Maintenance"
Communication of ACM, 21, 6 June 1978 pg 466 - 471

5. Daly E. B.
"Management of Software Development"
IEEE Transactions on Software Engineering SE-3 May 1977 pg 229 - 242

6. Isaacson P. and Juliussen E.
"Guset Editorial : Window On The 80's"
Computer Vol 13, 1 Jan 1980 pg 4 - 6

7. Lientz B. P. and Swanson E. B.
"Software Maintenance Management"
Addison-Wesley Publishing Co. Inc. 1980

CHAPTER ONE

SOFTWARE MAINTENANCE LIFECYCLE

1.0 INTRODUCTION

A typical graphical representation of the software life cycle is shown in figure 1.1

FIGURE 1.1

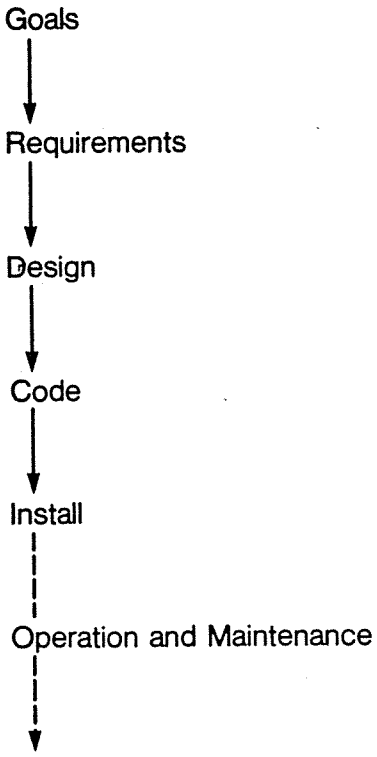


Figure 1—"Life cycle"

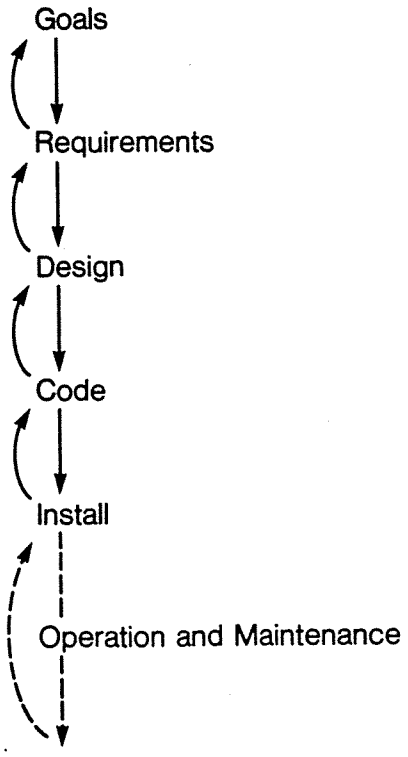


Figure 2—"Life cycle" with feedback

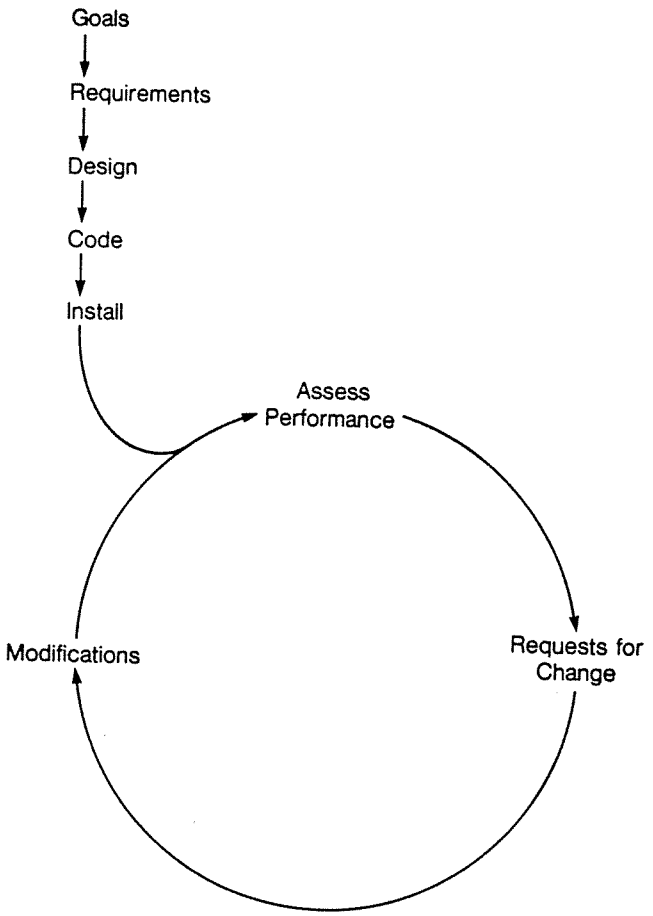
Adapted from Zvegintzov N. - Op Cit pg 563

This elaborated life cycle begins with problem recognition, then step through analysis, design, coding, installation, testing, operation and finally maintenance. Though the exact number of

phases and their names may vary between different authors(1), the basic structure remains very much the same. There is a serious deficiency with the above model, however. It fails to accurately depict the complete life software life cycle. As pointed out by Zvegintzov, the above model itself is not a cycle but rather a linear concept.

Zvegintzov(2) suggests that the real software life cycle consists of a mixture of linear and cyclical concepts as shown in figure 1.2

FIGURE 1.2



The starting gate model

Adapted from Zvegintzov N. - Op Cit pg 564

The above diagram clearly demonstrates that the software life cycle consists of two components. The first part involves the linear function of software development (ie from the goal identification phase to the installation phase). The second part is the software maintenance life cycle. The rest of this chapter is devoted to expanding the concept of this software maintenance life cycle and its importance in software maintenance management.

1.1 ELEMENTS OF THE SOFTWARE MAINTENANCE LIFE CYCLE (SMLC)

The maintenance process can be divided into the following 8 activities (3):

1. requesting phase
2. verification and estimation phase
3. scheduling phase
4. programming phase
5. testing phase
6. documentation phase
7. training phase
8. implementation phase
9. monitoring phase

1.1.1 Requesting Phase

The maintenance process begins when a user perceives a problem in the existing system or when an enhancement is called for. The user will then express this problem or enhancement as a change request and communicate this change request to the DP department via a change request form.

1.1.2. Verification and Estimation Phase

Upon receiving the change request form, a verification is done to establish the seriousness of the reported problem or the necessity of the requested enhancement. This is an important phase as it helps to filter out those problems that are a result of non-adherence to operational procedures, user misunderstanding, or invalid data. Once the problem/enhancement is certified to be genuine, an estimate is made of the number of hours to correct the error or incorporate the enhancement into the existing system. This estimate is then used as basis for the scheduling phase. Note that this is a decision point whereby DP management

must decide whether or not to implement the proposed changes.

1.1.3 Scheduling Phase

In order to handle the change requests efficiently, the change requests for a particular application are batched together to be implemented at a predetermined date. The concept of scheduled maintenance management(4) is discussed in detail in section 5.5.1.6 of this report.

1.1.4 Programming Phase

This phase of the maintenance life cycle is equivalent to the design and coding phase of the development life cycle. In the design stage, the maintainer attempts to map the problem as perceived by the user into a problem which is expressed in terms of the software(5). This is followed by a determination of what changes need to be made to the software. Next, the coding phase, concerns itself with modifying the existing program code to the specification derived in the design phase.

Note, however, that the programming phase in the maintenance life cycle could require more computing effort than the equivalent phase in the development life cycle. This is the result of additional activities that are required in the maintenance process but not in the development process. For example, in the maintenance process, action has to be taken to ensure that the modification is done on the current operational version of the system. Such action is not required in the development process.

After making the appropriate modification to the program code and the data representation, the modified program has to be checked and tested.

1.1.5 Testing Phase

The testing phase is the most crucial phase in the software maintenance life cycle. It ensures that the modified program satisfies the specification laid down in the design stage and that the modification to the program does not affect the integrity of the system environment.

The quality of testing is dependent on the thoroughness of the test plan and the quality of the test data(6). The test plan should be comprehensive and should incorporate unit testing, integration testing, and system testing of the changed elements and their interfaces.

The test data should encompass both good and bad data, expected and unexpected data. Regression testing should be done to ensure that of the program modification has no unforeseen side-effects.

The testing phase should not only involve DP personnel, but also the user group. For instance, program functionality can be evaluated by the user group. This not only contributes to confidence in system integrity but also helps in training the user using the modified application system.

1.1.6 Documentation Phase

Even though the official documentation update may begin at this stage, preparation for such documentation updating should be done in previous phases(7).

All documentation affected by the changes must be updated. This includes the system documentation, user documentation and run documentation. Detailed discussion of various documentation techniques is presented in chapter 5-5.2.8of this report.

1.1.7 Training Phase

This phase may not be necessary depending on the nature of the changes. For example if the changes involve modifying the application function then the user should receive appropriate training on how to use the enhanced function.

1.1.8 Implementation Phase

The implementation phase is the last step in the change process. An important point to note here is that the implementation must have a definite cutover schedule. Without such control, the system may never turn over to production but instead continue to run as test systems. This can lead to multiple versions and confuse users.

1.1.9 Monitoring Phase

The monitoring phase begins immediately the modified program is turned over to production. The monitoring phase plays an important role in the maintenance process in that it acts as an additional check on the integrity and reliability of the modified program.

1.2 IMPORTANCE OF THE SOFTWARE MAINTENANCE CYCLE CONCEPT TO MAINTENANCE MANAGEMENT

The SMLC concept is an important aid to maintenance management. It helps to reduce a complex process into its identifiable and measurable parts thus aiding in the assignment of resources, training of novices and specialization of roles(8). It acts as a significant step towards incorporating checkpoints in the maintenance process to ensure effective and efficient software

maintenance.

REFERENCES

1. Kerla P and P Freeman
   "A Comparison of Lifecycle Models"
   IEEE Computer Society 5th International Conference on Software Engineering 1981 pg 90-99

2. Zvergintzov N.
   "What Life? What Cycle?"
   AFIPS National Computer Conference June 1982 pg 561-568

3. Taute B.J.
   "Quality Assurance and Maintenance Application Systems
   AFIPS National Computer Conference 1983 pg 123 - 129

4. Lindhorst W.M.
   "Scheduled Maintenance of Applications Software"
   Datamation May 1973 pg 64-67

5. Harrison W., Magel K and Kluczny R.
   "Research in Software Maintenance"
   Journal of Systems Management July 1983 pg 12

6. Taute B.J. - Op Cit pg 129

7. Liu C.C.
   "A Look at Software Maintenance"
   Datamation November 1976 pg 51-55

8. Zvegintzov N. - Op Cit pg 567

CHAPTER TWO

SOFTWARE MAINTENANCE PROBLEMS

2.0 INTRODUCTION

Unlike hardware, software does not wear out, but its usefulness may diminish. Software can be compared to a child, which has to evolve the meet changing environments and needs. The usefulness of software diminishes with age however because of its inability to adapt itself to meet changing information requirements.

Software maintenance is not an easy task. It is beset by a host of problems, some of which are beyond the control of the organization concerned with software maintenance.

Lientz and Swanson (1) classified maintenance problems into 5 major categories :

1. user knowledge

2. programmer effectiveness

3. product quality

4. programmer time availability

5. system volatility

2.1 USER KNOWLEDGE

One of the major problems of software maintenance is lack of user understanding of the application system begin maintained(2). It is common for the user not to understand the cost and effort required to implement change requests ; nor does he know how to modify the change request to make it more feasible. This may result in the user making a continual stream of change requests to the Data Processing(DP) department without regard to their cost-effectiveness. As a result, a backlog of software maintenance may be worsened. Current research(3) shows that most companies have three to four years backlog of computer applications awaiting maintenance. The severity of this backlog is undoubtly aggravated by lack of user knowledge.

2.2 PROGRAMMER EFFECTIVENESS

Another problem facing software maintenance is the apparent misconception of DP management that the maintenance task is

generally easier than the development task. As such, maintenance activities are considered to need less planning, less expertise, fewer technical tools and less management direction. This mistaken belief often results in DP managers placing more experienced DP personnel on software development and less experienced personnel on the maintenance task. Some DP managers even go to the extent of treating the maintenance activity as a training ground for new employees. However , a simple maintenance job may turn out to be a difficult task for an inexperience programmer.

Contrary to this misconception, software maintenance activities are usually more difficult and consume more resources (programmer time) than software development. For example, when correcting a program error, the entire application system may need to be analyzed so as to determine which modules of the system contribute to the error. A decision has then to be made on how to modify the code to correct the problem. Finally, the maintenance changes need to be reassessed to determine the ramifications of the modification.

This situation is even worse when the error is not a trival diagnosable fault but rather a fundamental error in the design or implementation. This type of error may occur at any of a number of levels of program representation ranging from the problem specification down to the final implementation stage.

Furthermore, since the initial development of a system, it might have been modified and remodified to the extent that it may even reach a stage where there is little resemblance between the initial and the present version. Moreover, the people who design systems are usually long gone from the scene leaving the maintenance work to an inexperienced software maintainer(4).

2.2.1 Programmer Motivation

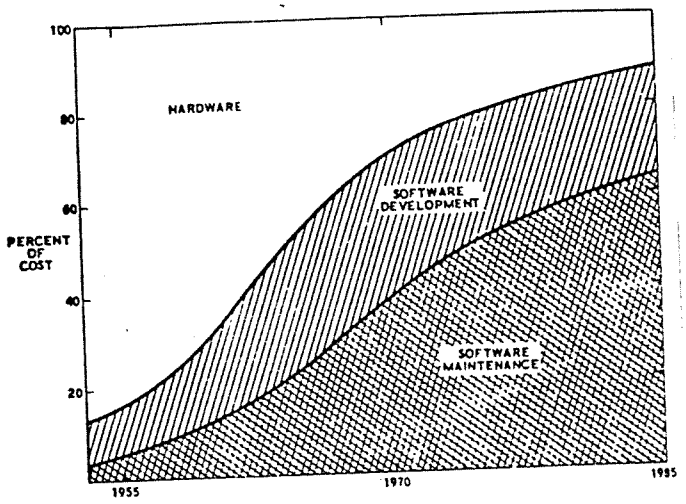
There is a stigma attached to software maintenance(5). Many people view software maintenance as non-challenging, unrewarding and only fit for beginner programmers. They consider maintenance work as "dirty" and done behind the scenes. To them the maintenance task has little glory, produces little progress and does not lead to promotion.

Maintenance programmers have a tendency to consider maintenance to be "unfair" because the culprit (the person who caused the maintenance problems) got promoted or left for a better job and left the "dirty work" to them with minimum assistance. They consider maintenance to be intellectually difficult as the cause of the problem can be anywhere. Some even equate maintenance with "repair", "fixing the bugs" and "pest control". These factors contribute to the creation of a sense of inferiority and frustration among software maintainers.

2.2.2 Programmer Time Availability

Maintenance, like other functions of the DP department, has grown rapidly in size and scope. It has in fact grown into a function that consumes most of the computing resources. One study showed that 75% of DP personnel are already taken up with maintenance(6). The increasing maintenance effort as compared with new development effort is shown in figure 2.1

FIGURE 2.1



Hardware–software cost trends.

Adapted from Boehm B.W. "Software Engineering"
IEEE Transaction On Computers Vol 25, 12 Dec 1976 pg 1227

In spite of the obvious importance of software maintenance, the maintenance activities have to compete with other D.P. functions (eg. new development) for programmer time. The dilemma here is that too few people have to undertake a task of ever increasing complexity.

The marketplace itself also creates competition for DP personnel, exerting pressure on the current maintenance staff to leave their current jobs. This is evidenced by the large number of advertisements for experienced programmers in the mass media.

2.2.3 Maintenance Management

Maintenance often has the outward appearance of being an unplanned, uncoordinated activity rather than a systematic process necessary for the business function of the D.P. organization(7). The problem does not relate so much to the technology as to the manner in which the maintenance activities

are carried out.

The tools and techniques for effective management are available but they are under-utilized and hence maintenance management is often undisciplined.

2.3 PRODUCT QUALITY

The third major problem facing software maintenance is that maintenance is difficult to undertake on a system which wasn't designed for maintenance in the first place. The adequacy of application specification and system design, and the quality of the original programming, play an important role in determining the subsequent difficulty of the maintenance task.

Many organizations still operate today on computerized applications developed during the sixties. Many of these older application systems do not have today's well structured system designs and weren't designed with maintainability in mind. Maintaining such poorly designed systems is both difficult and time consuming.

2.3.1 Documentation

Traditionally, the responsibility for documenting an application system rested upon the analyst/programmer who was involved in the development of the system. Documentation written by programmers is usually unsatisfactory because programmers are generally sloppy and disorganized about this aspect of their work, and often use the English language as though it were a programming language(8).

Moreover, documentation by DP personnel tends to be a personal and individualized activity and is often incomplete. As a result, system maintenance based on the available documentation is difficult, and for a person who was not involved in the system development, the maintenance task is made much more difficult.

2.3.2 Ripple Effect

The next problem with software maintenance is the possibility of introducing an unforseen error as the result of a software change, commonly known as the ripple effect(9). For instance, fixing a bug in one part of a program may itself create another new bug elsewhere in the program. This is especially so if the application program is not well-designed and well-documented, as such side effects cannot be anticipated until they cause problems during the operational stage. As a consequence, software maintenance often needs far more program testing per line than software development. Furthermore prolonged software maintenance on an application system can result in deterioration of its structure thus making it even more complex and difficult to

maintain next time round. This can result in more time being spent on fixing the symptoms than correcting the causes of the original problem.

2.4 SYSTEM VOLATILITY

Instability of the operating environment (both hardware and software) supporting an application system can pose a serious problem to the system maintainer. With the rapid advancement of computer technology, operating environments change rapidly. This creates more problems for the software maintainer in order to ensure the integrity of the operating environment.

The software maintainer must address these problems to achieve effective maintenance. The software maintainer must understand the factors contributing to these problems and then develop a set of maintenance strategies to counter the problems.

REFERENCES

1. Lientz B. P.
   "Software Maintenance Management"
   Addison-Wesley 1980 pg 115 - 148

2. Lientz B. P. and Swanson E. B.
   "Problems in Application Software Maintenance"
   Communication of ACM Vol 24 No. 11 Nov. 1981 pg 763 - 768

3. Martin J. and McClure C.
   "Software Maintenance : The Problem and Its Solutions"
   Prentice Hall N.J. 1983 pg 5 - 7

4. Parikh G.
   "Techniques of Program and System maintenance"
   Winthrop Publishers Inc. 1982 pg 53 - 55

5. Reutter J.
   "Maintenance Is A Management Problem and A Programmer's Opportunity"
   AFIPS conference proceedings 1981 May 7 pg 343 - 347

6. "Program Maintenance : User's View"
   Data Processing Sept - Oct 1983 pg 314 - 317

7. Reutter J. - Op Cit pg 343

8. Liu C. C.
"A look at Software Maintenance"
Datamation Vol 22 Nov 1976 pg 51 - 55

9. Yau S., Collonfello J. and MacGregor T.
"Ripple Effect Analysis on Software Maintenance"
IEEE 2nd International Computer Software and Application Conference Proc. Nov 1978 pg 60 - 65

CHAPTER THREE

SOFTWARE MAINTENANCE CLASSIFICATION

3.0 INTRODUCTION

A fundamental conceptual issue in software maintenance is the definition of the word maintenance .Riggs(1) defines maintenance as "the activity associated with keeping operational computer systems continuously in tune with the requirements of users, data processing operations, associated clerical functions and external demands from governmental and other agencies".

Ogin(2) differentiates maintenance from modification. He defines maintenance as "the continuing process of keeping the program running or improving its characteristics .....
and program modification has as one of its objectives adapation to a changing environment".

Mooney(3) states that maintenance consists of three activities: repairs, revisions and enhancenments. Canning(4) identifies four causes of software maintenence: program won't run , program runs but produces wrong results , business environment changes ,and finally enhancement and optimization.

Boehm(5) divides maintenance into software update and software repair. The difference between the two lies in that the former involves a change of functional specification whereas the latter leaves the functional specification intact.

Arnold(6) has a more inclusive notion of maintenance " .. a collection of activities that relates to correcting , adapting or perfecting software in productive use".

"Software" includes not only program code but also the specification and design documentation concerning the program code. "Correcting " software involves removing functional errors. "Adapting " software means installing enhancements ."Perfecting " software entails an improvement in processing efficiency or performance and changeability. "Productive use" implies that the software has been formally accepted and put into operation by its user.

Other conceptual issues regarding the definition of software maintenance concerns the extent to which an enhancement or repair of the software constitutes software maintenance or software development. Boehm(7) addressed this issue by quantifying software maintenance as :

1. redesign or development of existing software requiring less than 50% of new code

2. redesign and development of smaller interfacing software packages which require redesign of less than 20 % of the existing software

3. modification of the software code, documentation or data base structure

Any software maintenance that does not satisfy the above requirements is considered as new software development.

3.1 IMPORTANCE OF MAINTENANCE CLASSIFICATION

The first step in gaining a proper perspective on maintenance is to identify the various kinds of maintenance and to classify them. This is because different types of maintenance require different sets of maintenance strategies and maintenance skills. For instance, emergency maintenance usually involves correcting errors that otherwise would result in inhibiting part or the whole application system.This type of maintenance is likely to require the software maintainer to have an in-depth understanding of the operating system, file structure and network protocols of the application.

3.2 SOFTWARE MAINTENANCE CLASSIFICATION

The broad interpretation of the maintenance function outlined above has resulted in various ways of classifying the maintenance function.

Reutter(8) divided the maintenance function into 7 major classes: emergency maintenance, corrective maintenance, upgrade maintenance, change-in-condition maintenance, growth maintenance, enhancement maintenance, and supportive maintenance.

Perry(9) distinguishes the maintenance function into 7 main classes: corrective maintenance ,file maintenance ,emergency maintenance, deferred maintenance, preventive maintenance, scheduled maintenance and perfective maintenance.

Swanson(10) uses a similiar classification scheme but refines the classification into 3 groups namely ; corrective maintenance, adaptive maintenance and perfective maintenance.

1. Corrective Maintenance

Corrective maintenance is conducted for the purpose of identifying and correcting software failures, performance failures and implementation failures.

2. Adaptive Maintenance

Adaptive maintenance results from adapting current software to changes in the data processing environment, e.g. new hardware, operating system etc.

3. Perfective Maintenance

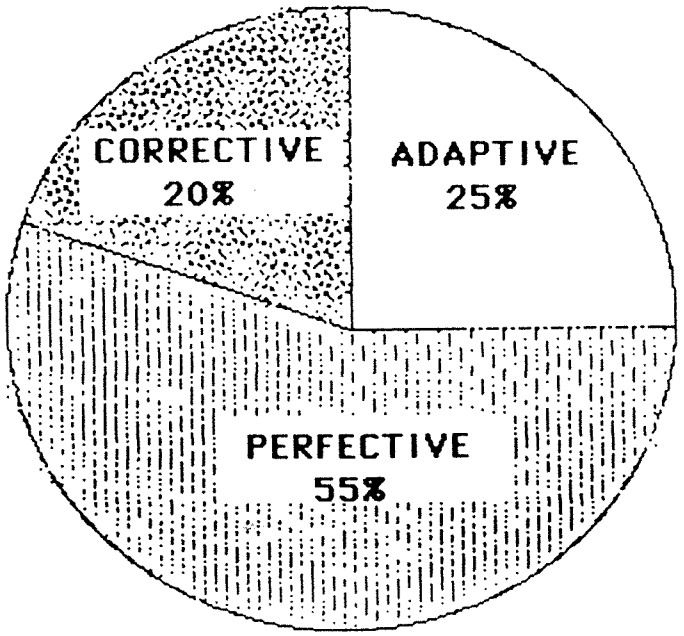
Perfective maintenance is performed to enhance software functions to meet changed user or external needs and to improve effectiveness, efficiency or maintainability of existing functions.

3.3 MAINTENANCE EFFORT BY CATEGORY

Reseach done by Lientz and Swanson(11) revealed in the late 1970's that the majority of maintenance falls within the category of perfective maintenance. This is clearly demonstrated in the piechart shown in figure 3.1

ALLOCATION OF COMPUTER RESOURCES TO MAINTENANCE TYPES

FIGURE 3.1

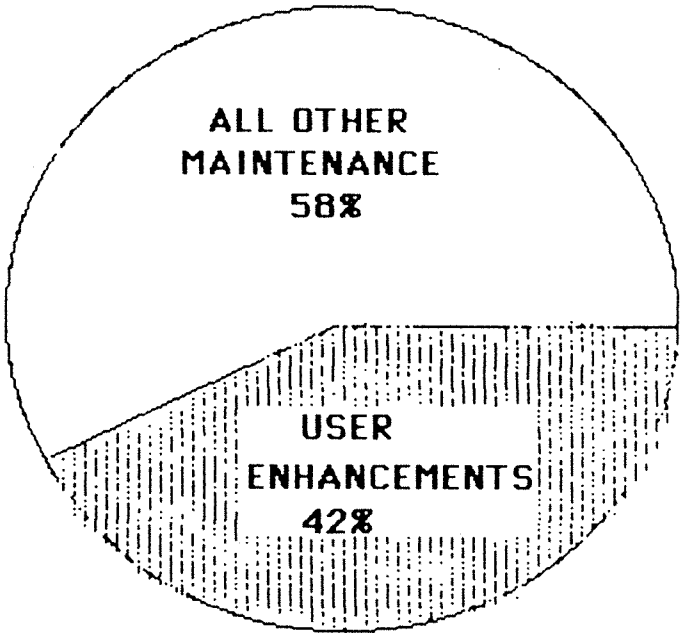


Data adapted from Lientz and Swanson
- Op Cit pg 151 - 157

From the piechart , it can be seen that perfective maintenance

accounted for over 50% of all maintenance effort , compared with 20% for corrective maintenance and 25% for adaptive maintenance. This result can be rather surprising to the DP community some of whom who have the misconception that the majority of the maintenance effort falls within the corrective maintenance category(12). The high percentage of perfective maintenance can be explained by the fact that user enhancement (a type of perfective maintenence) consumes almost half of all maintenance expenditure . This is depicted in figure 3.2

FIGURE 3.2



MORE INFORMATION	70%
REFORMATION	10%
CONSOLIDATION	10%
OTHER	10%

Data adapted form Lientz and Swanson
- Op Cit pg 151 - 157

Lientz and Swanson(13) in their study refined the classification of user enhancement to 4 types and noted that 70% of user enhancement is related to giving users more information rather than consolidating or reformatting existing information. See figure 3.2

For the purpose of this exploratory research, this report adopts Swanson's classfication of software maintenance but expands it to include supportive maintenance (defined below). Thus software maintenance is categorized into 4 main classes :

1. corrective maintenance

2. adaptive maintenance

3. perfective maintenance

   (For the definition of the above maintenance categories refer to Swanson's definition mentioned earlier).

4. supportive maintenance

   Supportive maintenance involves the explanation of the system capabilities, system outputs and proper use of system features to end users of the system(11). It provides assistance to end users in planning changes and support. Supportive maintenance also includes measuring and evaluating systems performance.

REFERENCES

1. Riggs R.
   "Computer Systems Maintenance"
   Datamation Nov 1969 pg 227 - 232

2. Ogdin J. L.
   "Designing Reliable Software"
   Datamation, July pg 71 - 78

3. Mooney J. W.
   "Organized Program Maintenance"
   Datamation Vol 21 Feb 1975 pg 63 - 66

4. Canning R.
   "The Maintenance Iceberg"
   EDP Analyser Oct79 pg 1 - 4

5. Boehm B. W.
   "Software Engineering Economics"
   Englewood Cliffs, NJ Prentice Hall 1981 pg 35 - 51

6. Arnold R. S.
   "The Dimensions of Healthy Maintenance"
   IEEE 1982 0270 - 5257 pg 10 - 27

7. Boehm B. W. - Op Cit pg 35 - 51

8. Reutter J.
"Maintenance Is A Management Problem and A Programmer's Opportunity"
AFIPS Conference proceedings 1981 May 7 pg 343 - 347

9. Perry W. E.
"Managing Systems Maintenance"
Q.E.D Information Sciences Inc.
Wellesley Massachusetts 1981 pg 1 - 12

10. Swanson E. B.
"The Discussions of Maintenance"
2nd International Conference on Software Engineering
proceedings October 1976 pg 492 - 497

11. Lientz B. P. and Swanson E. B.
"Software Maintenance Management"
Addison-Wesley Publishing Co. Inc. 1980 pg 151 - 157

12. Martin J. and McClure C.
"Software Maintenance : The Problem and Its Solutions"
Prentice - Hall NJ 1983 pg 31 - 33

13. Lientz B. P. and Swanson E. B. - Op Cit pg 151 - 157

14. Martin J. and McClure C. - Op Cit pg 28

CHAPTER FOUR

FACTORS AFFECTING SOFTWARE MAINTENANCE

4.0 INTRODUCTION

Strategies for effective software maintenance cannot be formulated without a thorough understanding of the factors affecting it. These software maintenance factors can be divided into 2 classes: external factors and internal factors(1).

External factors are those whose which relate to the environment of the software. These include staff stability, programmer quality, frequency and extent of maintenance, user sophistication and hardware stability.

On the other hand , internal factors have an impact relating to the design of the software. The internal factors include module independence, programming language, programming style, program testability, program documentation, program understandability, database management system interface and program complexity.

4.1 EXTERNAL FACTORS

4.1.1 Staff Stability

It is often easier for the programmer who has participated in the development of the software to maintain the software than someone else who has little or no knowledge of the software. This is because the learning curve on the application system is relatively short for the original writer of the program compared with another individual who has to understand the application program by studying its documentation. Depending on the previous experience of this other individual on the application program, the learning process can be both time consuming and unproductive.

However, in the real world, it is not a common practice for the software developer to maintain the software he developed. This is because programmers tend to job hop frequently. A study done by Lientz(2) revealed that for some organizations, the DP staff turnover rate is as high as 30% per annum. Some recent NZ experience indicates even higher turnover in many organizations. This is particularly so for maintenance staff.

4.1.2 Programmer Quality

The quality of the programmers who developed the software is a significant factor affecting the program's subsequent maintenance. The higher the quality of the programmer, the less

likelihood of semantic errors, and the greater likelihood that the software will be thoroughly tested before implementation. Research done by Endres(3) and Weinberg(4) showed that there is a significant positive correlation between the quality of the programmer and the amount of subsequent maintenance needed by a module within the operating system that they studied.

From a maintenance point of view, programmer quality can be characterized as (5)

1. programmer experience with the application program presently being maintained

2. programmer "innate" ability

3. programmer motivation

4.1.3 Frequency and Extent of Maintenance

The frequency and extent of repairs is another factor that can affect future maintenance.

Software which experiences frequent modification is difficult to maintain because of the likelihood of out-dated documentation. Furthermore, frequent maintenance can result in complicating the logical flow of the program structure.

A study done by Boehm(6) disclosed a strong relationship between the size of the repair and the maintenance function. His study revealed that with a minor modification, involving changes of less than 10 statements, the chances of a successful first run after the modification are approximately 50%. However, for changes involving 50 or more statements, the chances of a successful first run dropped to below 20%. The same conclusion was derived by a study done by Vessey and Weber(7).

4.1.3.1 The dependence of the program on its external environment

A program that is highly dependent on its external environment will usually require more maintenance throughout its useful life compared with software which has a low dependence on its environment.

For example, with the introduction of goods and services tax in 1986, invoicing application programs have to be modified to include taxes in the calculation of sale prices.

However, such changes in the taxation legislation may have little impact on modules such as accounts receivable application programs, or inventory control.

4.1.4 User Sophistication

User sophistication is characterized by how easily the user can identify the software problems or perceived needs, how accurately the user can interpret the problems or needs and how well the user can communicate these needs and problems to the analyst/programmer.

A high degree of user sophistication is likely to result in lower perfective maintenance. This is because the program specification does not only satisfy the present needs more completely but also attempts to cater for future requirements.

This effect of user sophistication on the maintenance process is supported by a study done by Lientz and Swanson(8). In their study, they discovered that among the 26 different software maintenance problems identified, inadequate user training and lack of user understanding were considered among the most serious .

User sophistication is influenced by the following factors(9):

1. user's knowledge of the application area

2. user's knowledge of the computer software

3. diversity of the user community

4. diversity of applications addressed by the software

4.1.5 Virtual Machine Volatility

Virtual machine volatility refers to the frequency of changes in the hardware configuration, operating system, languages, compilers, linkers/loaders, DBMS, utilities, tools etc. that supports the application system.

Hardware instability will result in high adaptive maintenance because the application programs have to be modified to meet the new hardware configuration requirements.

4.2 INTERNAL FACTORS

4.2.1 Module Independence

Module independence can be defined as the ability to modify one program unit (module) of the application system without affecting other units (modules).

Programs written in a modular fashion should require less corrective maintenance because the modular design permits application program to be tested more thoroughly before implementation. Furthermore, by breaking a large application

program into small manageable modules, it tends to reduce program complexity and improves understandability.

An empirical study done by Vessey and Weber (10) concluded that there is a significant correlation between program modularity and corrective maintenance expenditure.

4.2.2 Programming Language

The type of programming language used to code the program has an impact on future maintenance effort. Programs written in high level languages are usually easier to understand and hence easier to maintain compared with those coded in low level languages (eg. assembly language).

Ancecedotal evidence gathered by Guimaraes (11) supports the existence of a relationship between type of program language and maintenance effort.

4.2.3 Program Testability

Program testability is concerned with how easily a program can be tested. It includes the ease of organizing the minimal number of tests that will satisfy testing criteria.

Programs with good program testability are expected to need less corrective maintenance.

Major factors that influence program testability include(12) :

1. availability of support and testing tools

2. availability of the test data

3. program modularity

4. program complexity

4.2.4 Programming Style

Good program design and coding enhance program understandability and hence ease of maintenance. Unstructured program design and coding deceases program understandability and hence increases the maintenance effort.

The importance of programming style is evidenced in Canning´s study(3). His findings revealed that the products of structured programming need less corrective maintenance than unstructured programs.

4.2.5 Program Documentation

The quality of program documentation has a great influence on the maintenance process. Programs supported by clear, comprehensive, and yet succinct, documentation are easier to understand, and hence the task of maintenance is eased. Consequently, programs with high quality documentation tend to have lower maintenance expenditure compared with those supported by poorly organized and incomplete documentation.

A recent study done by Guimaraes(14) concluded that there is a significant reciprocal relationship between the quality of the documentation and the average yearly maintenance expenditure.

4.2.6 Program Understandability

Program understandability refers to how easily a program or program module can be understood. Program understandability can be characterized as the ease with which the revelant functional modules can be isolated so that the problem can be easily identified, ie. the ease with which the needed changes can be determined and implemented.

Factors affecting program understandability include:

1. complexity in the control flow structure

2. complexity in the data flow structure

3. the quality of the documentation

4. program modularity

5. previous maintenance of the software

Harrison, Magel and Khuczny(15) suggested that maintenance effectiveness can be increased by improving program understandability.

4.2.7 Database Management System Interface

Application programs using a database management system tend to have a lower maintenance expenditure than those not involving database interface. This is attributed to the fact that database management systems (DMS) provide data independence and hence result in the data files becoming less sensitive to changes in the information processing requirements. This reduces the future perfective maintenance effort. Guimaraes(16) did an extensive study of this area and arrived at the conclusion that programs with DMS interfaces tend to have a lower average maintenance expenditure per year per line of code compared with those without DMS interfaces.

4.2.8 Program Complexity

There is a direct relationship between program complexity and software maintenance expenditure(17). Complicated programs are difficult to understand and hence difficult to maintain.

Program complexity can be characterized by:

1. the number of interfaces between the application subsystems

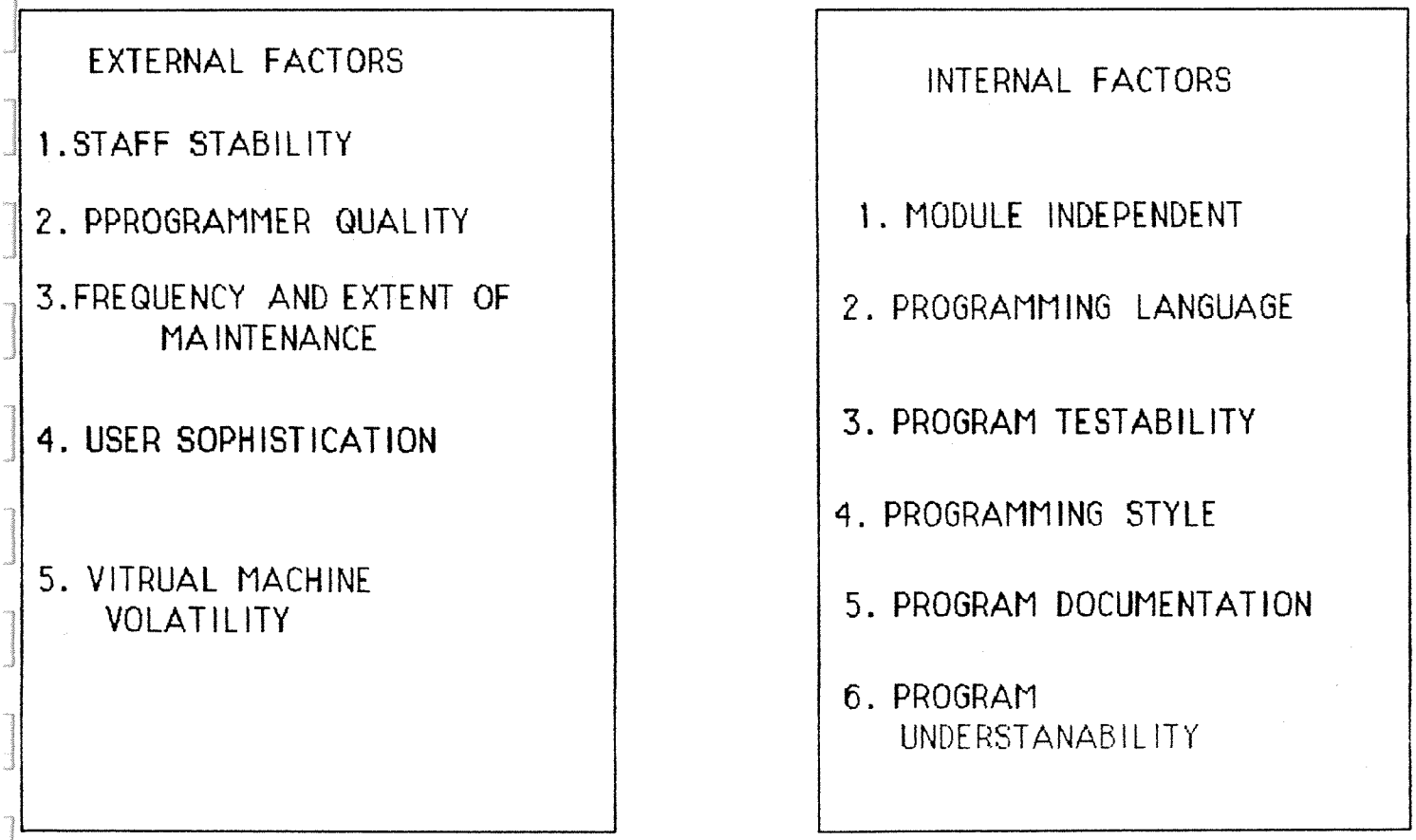
2. the degree of software interaction with the hardware

3. the complexity of the data and program structures

Factors affecting the maintenance process can be summarised in the diagram shown in figure 4.1

FIGURE 4.1

FACTOR AFFECTING MAINTENANCE PROCESS



REFERENCES

1. Sommerville I.
"Software Enginerring"
Internatiomal Computer Science Series
Addison-Wesley Publishing Co., 1982, pg 198-203

2. Lientz B. and Swanson E.
"Problems In Application Software Maintenance"
Communications of ACM, Nov 1981, pg 763-769

3. Enders A.
"An Analysis of Errors and their Causes in System Programs"
SE 1,2, June 1975, pg 140-149

4. Weinberg G. M.
"The Psychology of Computer Programming"
Van Nostrand Reinhold, N. Y., 1971

5. Harrison W., Kenneth M. and Klucznv R.
"Research in Software Maintenance"
Journal of Systems Management, July 1983, pg 10-14

6. Boehm B. W.
"Software and Its Impact : A Quantitative Assessment"
Datamation, 19,5; May 1973, pg 48-5

7. Vessey I. and Weder R.
"Some Factors Affecting Program Repair Maintenance : An Empirical Study"
Communications of ACM, Vol 26, Feb 1983, pg 128-134

8. Lientz B. P. and Swanson E. B.
"Software Maintenance Management"
Addison-Wesley, 1980, pg 115-148

9. Harrison W., Mogel K. and Kluczny R. - Op Cit pg 13

10. Verssey I. and Weder R. - Op Cit pg 129

11. Guimaraes T.
"Managing Application Program Maintenance Expenditures"
Communications of ACM, Oct 1983, pg 739-745

12. Harrison W., Magel K. and Kluczny R. - Op Cit pg 14

13. Canning R. G.
"Modular Cobol Programming"
EDP Analyer, 10,7; July 1972, pg 1-14

14. Guimaraes T. - Op Cit pg 74-72

15. Harrison W. Magel K. and Kluczny R. - Op Cit pg 13

16. Guimaraes T. - Op Cit pg 742

17. Thayer T. A. , Lipow M and Nelson E. C.
"Software Reliability"
North Holland Amsterdam, 1978

CHAPTER FIVE

FRAMEWORK FOR SOFTWARE MAINTENANCE MANAGEMENT

The task of setting up a software maintenance management program is complicated by the lack of a reliable model of the maintenance process.

For the analytical purposes of this study , a conceptual framework of the maintenance function is constructed by logically combining and arranging the various maintenance activities that appear to be reasonable parts in an organized effort to cope with the maintenance function.

This conceptual model is illustrated in figure 5.1

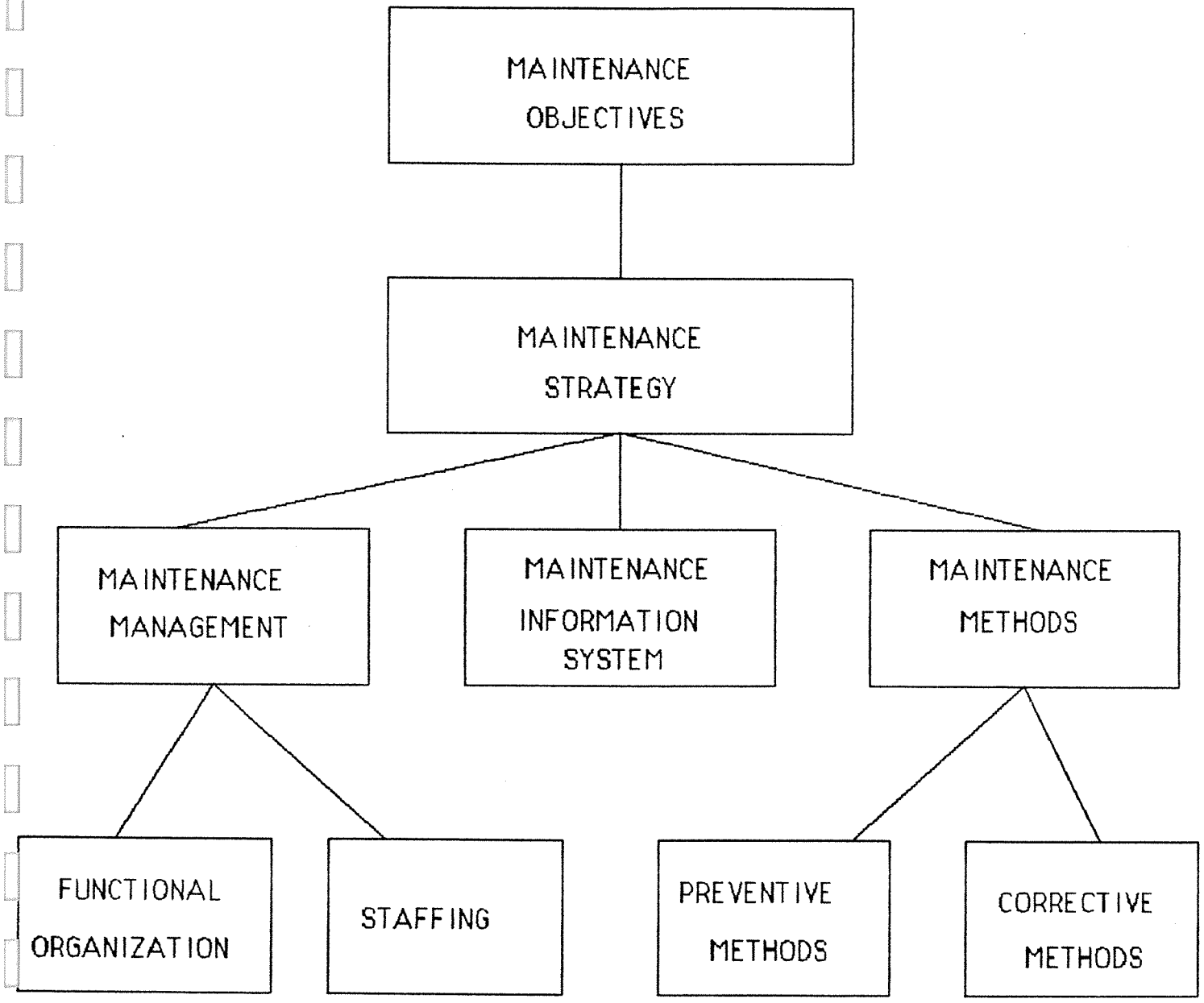
The framework can be categorized into the following 6 elements :

1. software maintenance management objectives
2. software maintenance management strategy
3. software maintenance measurement methodoloy
4. software maintenance management information system
5. software maintenance management methods
6. softwate maintenance functional organization

The elements of this framework are analysed in chapter 5.1 to chapter 5.6

FIGURE 5.1

FRAMEWORK FOR SOFTWARE MAINTENANCE MANAGEMENT



5.1 SOFTWARE MAINTENANCE OBJECTIVES

5.1.0 INTRODUCTION

The objectives of software maintenance are not as straight forward as one might expect. The goal of fixing errors is only one of the many objectives effective software maintenance should strive to achieve. There are other goals eg. software maintainability and software efficiency that the maintenance function must consider.

Software maintenance activities tend to vary from company to company which can complicate the objectives further. Moreover, there is a wide gap between management's and the maintainer's perception of the maintenance function. DP management is always demanding better results for less resources while the maintainer claims that the resources are too tight and hence impinge on software maintainability. The only way out of this impasse is to have the maintenance objectives defined in clear terms and to incorporate these objectives within the Information Systems strategy. In this chapter, an attempt is made to identify the general objectives of software maintenance and to rank them according to their relative importance. These objectives are :

1. software reliability

2. software correction

3. software modifiability

4. software maintainability

5. software efficiency

5.1.1 SOFTWARE RELIABILITY

The standard overall objective of a maintenance strategy should be the maintenance software reliability.

A program is considered reliable if it satisfies the following criteria (1) :

1. meets its specification

2. always produces "correct" output irrespective of the input

3. never allows itself to be corrupted

4. takes meaningful and useful actions in unexpected situations

5. completely fails when further progress is completely

impossible

In short, software reliability is the ability of the user to use the software and get correct results.

However, note that software reliability does not equate with "absolute" correctness. To set an ideal objective of achieving a system that satisfies all perceived and projected needs is both impractical and unrealistic. Rather, as Lehman(2) points out, "it is the usability of the program in the real world and the relevance of its output in the constantly changing environment that should be the main concern in achieving software reliability".

5.1.2 SOFTWARE CORRECTION

In general error correction should be the next goal in the list of maintenance objectives(3). When an error is discovered in the application program, it is usually important to fix it as soon as possible unless the nature of the error is such that the cost of fixing the error exceeds the value of fixing it.

The priority of error fixing should be relative to the seriousness of the error itself. Errors that can completely halt the operation of the system should have the highest priority while those that have a very low probability of occurence should have the least priority.

5.1.3 SOFTWARE MODIFIABILITY

The third objective of a software maintenance function is to satisfy the change requests of the user(4).

In general, an error correction should be given a higher priority than a change request. However, a high-priority change request could be done before low-priority error fixing. The relative priorities between error fixing and change requests should be established in principle before the development of the maintenance strategy.

5.1.4 SOFTWARE MAINTAINABILITY

It is often difficult to convince top management that a specific modification/enhancement is necessary, especially when it does not result from a specific problem or change request(5).

Restructuring a disorganized, undisciplined program or generalizing a special-purpose program or annotating unreadable documentation enhances software maintainability(6).

Improving software maintainability will reduce the overall maintenance expenditure and in some cases might even result in

lengthening the useful life of the software.

However, this objective must not be over-emphasized. A balance must be kept between software maintainability and the company's available computer resources.

5.1.5 SOFTWARE EFFICIENCY

Software consumes computer resources - storage space, computer time, programmer time, and user time. Therefore, improving software efficiency is another valid software maintenance objective(7). But this objective should not be carried to the extreme. Efficiency concerns depend on the type of software. Efficiency is important for heavily used software eg. operating systems, but for many typical data-processing systems efficiency should not be achieved to the detriment of other software maintenance objectives (eg. maintainability).

Once the software maintenance objectives have been identified, the DP manager is in a position to develop a software maintenance strategy.

REFERENCES

1. Sommerville I.
   "Software Engineering"
   International Computer Science Series
   Addison-Wesley Publication Co. 1982 pg 198 - 203

2. Lehman M. M.
   "Program, Life Cycles, and the Laws of Software Evolution"
   Proc. IEEE 1980 pg 1060 -1076

3. Glass R. L. and Noiseux R. A.
   "Software Maintenance Guidebook"
   Prentice - Hall 1981 pg 33 - 34

4. Glass R. L. and Noiseux R. A. - Op Cit pg 33.

5. Glass R. L.
   "Software Reliability Guidebook"
   Prentice-Hall 1979

6. Glass R. L. and Noiseux R. A. - Op Cit pg 34

7. Glass R. L. and Noiseux R. A. - Op Cit pg 33 - 34

5.2 SOFTWARE MAINTENANCE STRATEGY

5.2.0 INTRODUCTION

Just like any other DP function, the maintenance process needs strategic planning. Anthony(1) defines strategic planning as the process of "deciding on the objectives of the organization, on changes in these objectives , on the resources used to attain these objectives, and on the policies required to govern the acquisition , use and disposition of those resources".

A well-developed strategic plan for the software maintenance function is essential for a consistent and effective software maintenance framework. It reduces conflicts between maintenance activities and results in more productive use of scarce resources. Many of the expensive aspects of maintenance are the result of failure to develop a well thought out maintenance strategy(2). The objectives of strategic planning in software maintenance are : (3)

1. to enable the DP organization to pursue its software maintenance objectives (see section 5.1 of this report)

2. to ensure that the software maintenance strategy developed is consistent with the overall information systems and corporate strategies

3. to ensure that the maintenance function is responsive to continuing changes in the DP environment

5.2.1 STEPS IN DEVELOPING A MAINTENANCE STRATEGY

Developing a software maintenance strategy involves 4 steps :

1. analysis of corporate missions and strategies

2. evaluation of current key applications through a well developed software maintenance information system

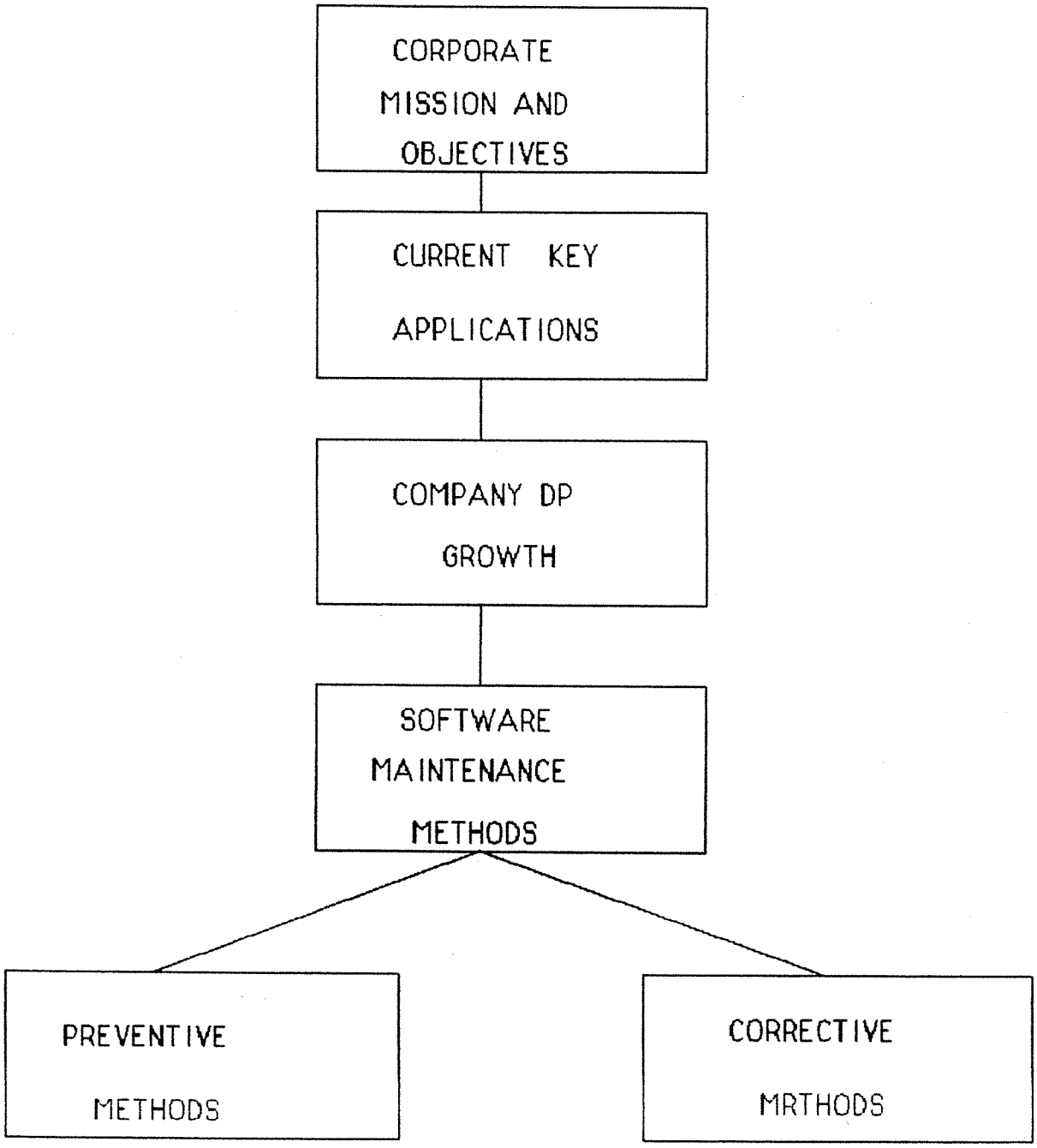
3. identifying the company´s present stage of DP growth

4. devising key software maintenance management techniques to improve maintenance productivity

This 4-step-approach to development of maintenance strategy can be depicted in figure 5.2.1

FIGURE 5.2.1

SOFTWARE MAINTENANCE STRATEGY DEVELOPMENT



5.2.1.1 Analysis of Corporate Objectives and Strategies

The overall corporate objectives and strategy for an organization have an important influence on the DP function of the company. Without a thorough understanding of the overall corporate strategy, the DP manager cannot develop an effective maintenance strategy and methods to control maintenance costs. To illustrate this vital point, assume that a corporate objective is to achieve corporate growth and the corporate strategy to attain this objective is through diversification. This means that many of the corporate application programs will be likely to change in the near future. The manufacturing control system, the accounting information system, manpower planning system and the decision support system would need to be enhanced to cater for new market segments and new subsidiary companies acquired by the organization. Therefore the software maintenance strategy developed should ensure that future enhancement maintenance of the existing system can be carried out smoothly. This implies that software maintenance activities and resources should be geared towards enhancement maintenance. The maintenance strategy is part of the overall information systems strategy.

5.2.1.2 Evaluate the Status of Key Applications

Pareto's principle states that 80% of the effort is spent on 20% of the task. Pareto's 80/20 rule is applicable to software maintenance. A study done by Weinberg(4) revealed that in general 80% of the maintenance effort is spent on maintaining less than 20% of the application system. The implication of this 80/20 rule on software maintenance strategy development is that the strategy developed should place its emphasis on the most frequently maintained software or maintenance category. This is important because of the limited amount of computer resources allocated to software maintenance. Streamlining the maintenance effort to concentrate on the most frequently maintained applications or maintenance categories will result in higher maintenance productivity. To apply this worst-first maintenance strategy(5), DP management must have adequate data on existing application systems. This can be achieved by setting up a software maintenance information system. The data to be collected and analysed by this system include : (6)

1. effort spent on maintaining the application program each year

2. program age

3. programming languages in which the program is coded

4. program error information

5. program change information

6. user satisfaction survey

7. maintainer opinion of the existing system

8. conformance of the application system to corporate data models

9. difficulty of interfacing the program with other systems

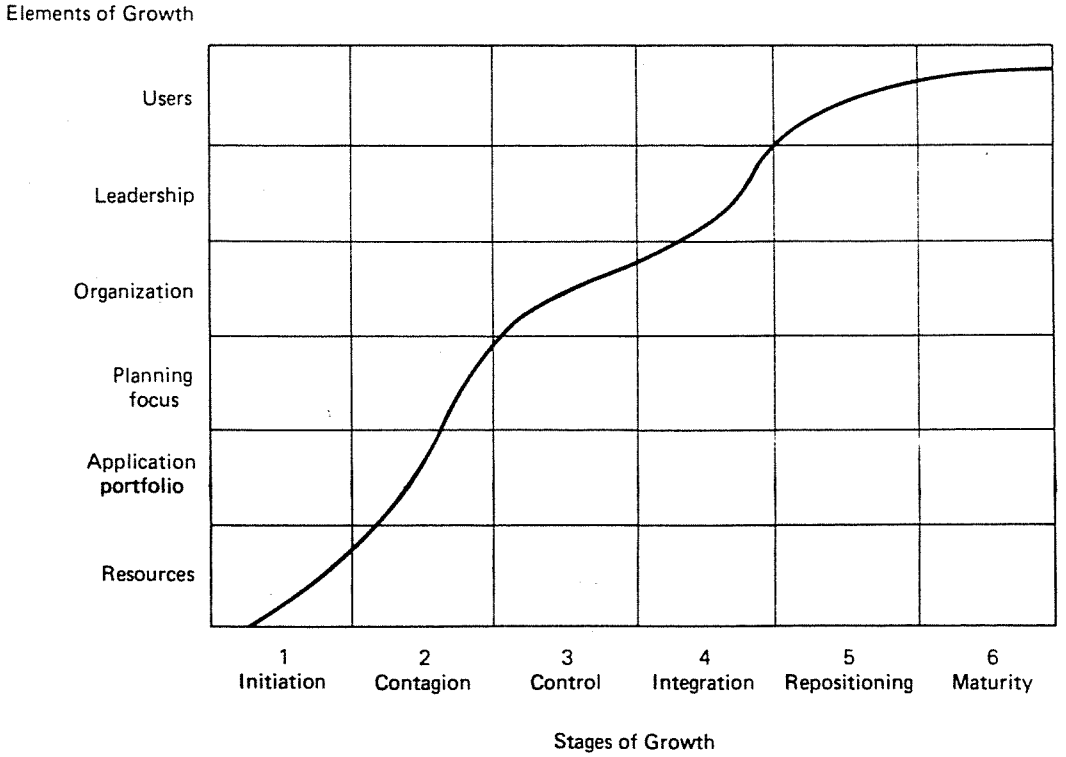
A more detailed analysis of the software maintenance information system is found in the next chapter of this report.

5.2.1.3 Identifying the Company's Present Stage of DP Growth

Understanding the DP growth stages of organisations and identifying the current stage of this process in a particular organization would provide many valuable insights into the magnitude and type of maintenance effort required in the near future. Such information is a vital input for the development of an effective maintenance strategy and maintenance techniques.

Nolan(7) divided the DP growth of a typical company into 6 main stages. This is shown in figure 5.2.2

FIGURE 5.2.2



The stages theory (Adapted from Richard Nolan's stages theory)

Stage 1, the initiation stage , commences with the company's purchase of a new computer system. The maintenance activities at this stage are minimal as the bulk of the computer resources are consumed in the software development process.

Stage 2, the contagion stage , is characterized by a rapid proliferation of applications, usually with little management control and awareness. The maintenance activities, mainly corrective and enhancement maintenance, begin to increase progressively as more and more application features are added to the system.

Stage 3, the control stage, involves increased management control and awareness on the application system. At this stage, planning, controlling and organising for future growth are emphasized. This stage also introduces the implementation of a steering committee. Maintenance costs (usually corrective and enhancement) are usually high. This is due to the undisciplined development of software arising from the previous two phases.

Stage 4, the redesign stage, begins with a complete overhaul of the design. for existing application systems using database technology. This stage also introduces the concept of an integrated database and network approach. Maintenance effort at this stage usually centres on adaptive maintenance.

Stage 5, the data administration stage, sees the introduction of data administration and the main focus is on planning of data resources and data models.

Stage 6, the maturing stage occurs when the functional systems are automated and the data structure reflects the organizational and information flow in the company. The maintenance activity begins to ease compared with the previous stages. A substantial amount of the maintenance effort falls within the category of supportive maintenance ie. providing information to the user for user maintenance activities.

5.2.1.4 Devising Key Software Maintenance Management Techniques

Once the strategic planner has adequate information concerning the company DP function, he is in a position to devise software maintenance management techniques. Basically, the software maintenance management techniques can be divided into 4 main classes, according to maintenance category.

The 4 main classes are :

1. corrective maintenance management techniques

2. adaptive maintenance management techniques

3. perfective maintenance management techniques

4. supportive maintenance management techniques

For detailed discussion of the software maintenance management techniques see chapter 5.5 of this report.

5.2.2 TOP MANAGEMENT INVOLVEMENT IN STRATEGIC PLANNING

For the software maintenance strategy to be effective, it must involve top management(8). This is because the software maintenance process often does not only involve the DP department but also other corporate functions. Without top management support, conflicts between the DP department and other corporate departments may result. For instance the chief accountant may be reluctant to expand his responsibility to include training of his accounting staff in the new role of participating in maintaining the accounting application system even though it is beneficial to the company as a whole.

Furthermore, top management are responsible for steering the company into the future. They are well versed in the enterprise's mission and strategy. This knowledge is vital to the development of a software maintenance strategy , as any conflict between the overall corporate strategy and the maintenance strategy will cause the latter to be less effective.

REFERENCES

1. Anthony R. N.
   "Planning and Control Systems : a Framework for Analysis"
   Graduate School of Business Administration, Harvard University
   1965

2. Martin J. and McClure C.
   "Software Maintenance : the problems and its solutions"
   Prentice-Hall Englewood Cliffs 1983 pg 453-465

3. Anoff H. I.
   "The Concept of Strategic Management"
   The Journal of Business Policy NO. 4 1972 pg 2-7

4. Weinberg G.
   "Worst-First Maintenance"
   in Techniques of Program and System maintenance
   Lincoln N E : Ethnotech 1980 pg 119-121

5. Weinberg G. - Op Cit pg 119-121

6. Martin J. and McClure C. - Op Cit pg 441-442

7. Nolan R. L.
"Managing the Crises in Data Processing"
Harvard Business Review Mar-April 1979 pg 115-126

8. Martin J. and McClure C. - Op Cit pg 463-464

5.3 SOFTWARE MAINTENANCE INFORMATION SYSTEMS

5.3.0 INTRODUCTION

Like any other area of the DP function, software maintenance requires information to enable optimum decisions to be made.

The objectives of a software maintenance information system are:(1)

1. to identify the factors that have impact on the maintenance process

2. to collect data relevant to the maintenance process

3. to convert the data collected into information necessary for effective maintenance decisions

4. to focus management attention on the importance of software maintenance

A software maintenance information system can be divided into 2 components: a maintenance identification system and a maintenance management information system.

The maintenance identification system supplies DP management with information such as installation and change request reports while the maintenance management information system provides details of the parameters and constraints which are vital to maintenance management decisions.

5.3.1 SOFTWARE MAINTENANCE IDENTIFICATION SYSTEM

The 2 strategies involved in developing a maintenance identification system are:(2)

1. analysis strategy

2. data collection strategy

Analysis Strategy

The first step in setting up any identification information system is to decide on the information to be collected and its use in the decision making process. The analysis strategy for a general identification system involves two elements(3) :

1. Statistical analysis of the system maintenance environment relating to the type and quality of maintenance performed. These statistics are a useful indication on the amount of

future maintenance needed for a particular application program.

2. Organizational analysis of the maintenance environment relating to its functional strengths and weaknesses. The strengths and weaknesses are related to the functional effectiveness of the maintenance organization (ie. maintenance team design) and to the DP organization as a whole. This information is a useful guide to setting up future maintenance project teams and the maintenance function.

Data Collection Strategy

The data to be collected by the maintenance identification system should be future-oriented since the aim of the maintenance information system is to make decisions regarding future maintenance. This means that both historical and forecast maintenance data are needed.

Historical maintenance refers to that maintenance that has already occurred, while forecast maintenance is maintenance waiting to be implemented. Perry(4) differentiates these 2 types of maintenance by suggesting that any maintenance that has reached the testing phase is considered as historical maintenance, while that not yet ready for the testing phase is categorized as forecast maintenance.

These two classes of maintenance data can be captured by using the installed change form.

Installed Change Form

A specimen copy of the installed change form is shown in figure 5.3.1. The installed change form is used every time a change request is initiated.

FIGURE 5.3.1

APPLICATION SYSTEMS									
CHANGE NUMBER	CHANGE DESCRIPTION	DATES		MAINTENANCE		MANHOURS		CODE	
		REQUEST	INSTALL	TYPE	CAT	BUDGET	ACTUAL	ORG	IMPT
COMMENTS									
PREPARED BY									

ADAPTED FROM PERRY W. E.- OP CIT PAGE 36

The main features of this form are the breakdown of the information into its maintenance type (ie. adaptive, perfective, corrective and supportive) and its relative importance. These features highlight the frequency of each maintenance type which is an important input into the software maintenance planning process.

Other features of the installed change form include :

1. Change Number

This is the number identifying to the application system to be changed and the specific change to the system.

2. Change Description

A brief narrative description of the change to been made.

3. Dates

The date when the change request was made and the date the request was implemented.

4. Maintenance Type

The code for maintenance type (ie. perfective, corrective, adaptive or supportive maintenance).

5. Maintenance Priority

This is the priority code used to decide the urgency for installing the change. Eg. code "1" use for critical maintenance and code "7" use for optional maintenance.

6. Hours of Effort

This includes both the budgeted hours and the actual hours of effort consumed in installing the change request.

7. Originator Code

The initiater of the change request. A typical originator code might be :

Code 1 - senior management
2 - auditor
3 - user
4 - DP personnel
5 - operations
6 - others

The installed change form should be completed by the project leader upon the completion of each maintenance process.

Self-Assessed Questionnaire

A tool that can be used to capture data necessary for organizational analysis is the self-assessed questionaire. The objective of the self-assessed questionaire is to provide DP management with an assessment of the adequacy of the maintenance function and its organization.

The aspects of software maintenance included in such a questionaire could be :

1. the validity of the software maintenance objectives

2. the degree of consistency between the software maintenance strategy and the corporate strategy

3. the effectiveness of the software maintenance management techniques currently practised by the DP department

4. the weakness of existing maintenance functional organization

The questionnaire should be completed periodically (e.g. yearly) by the DP manger, who is familiar with the organization's maintenance function. Based on the findings from the self-assessed questionaire survey, the DP management can identify the current and potential weaknesses of the maintenance function and its organization. They can then initiate further investigation to determine the seriousness of the problem, and implement necessary corrective action.

Note that the information gathered need not be 100% accurate so long as it is representative of the functional organization and the maintenance environment.

Software Measuring Aids

There are a number of software maintainability measuring tools in the commercial market. For examples, structure checkers and execution path tracers. Discussion of these tools and their usage in the maintenance process in presented in Section 5.5.1.1 of this report.

5.3.2 MAINTENANCE MANAGEMENT INFORMATION SYSTEM

Once the maintenance data has been identified via the maintenance identification system, the next step is to decide what action should be taken. The sequelae to the identification of the maintenance data require a broad understanding of the various factors that may influence the maintenance activities and hence the derivation of a feasible maintenance plan.

The maintenance data gathered can be analyzed in the following ways (6) :

1. maintenance analysis by application

2. maintenance analysis by maintenance type/category

3. trend analysis by number of change request

4. trend analysis by size of change

5. comparison analysis among application program

6. financial analysis using budgetting techniques

Financial Analysis

The most frequently used financial analysis technique for the software maintenance is budgetting.

The maintenance budget (measured in man hours) is a plan against which the ensuring actual performance is compared, so as to detect and correct non-standard maintenance performance. The difference between the planned and actual use of maintenance effort (manhours) is known as the maintenance variance. This variance is subjected to a significance test (eg. 10%) and any variance that fails the test will be investigated further for any maintenance inefficiency.

Note however, the ability to keep within the budget standard does not necessarily mean good maintenance performance particularly if such performance is achieved at the expense of the long term maintainability of the software. However, such a budget report is one of the many indicators for determining the maintenance efficiency and is an important piece of information for software maintenance manpower planning.

A specimen copy of the maintenance budget report is shown in figure 5.3.2

FIGURE 5.3.2

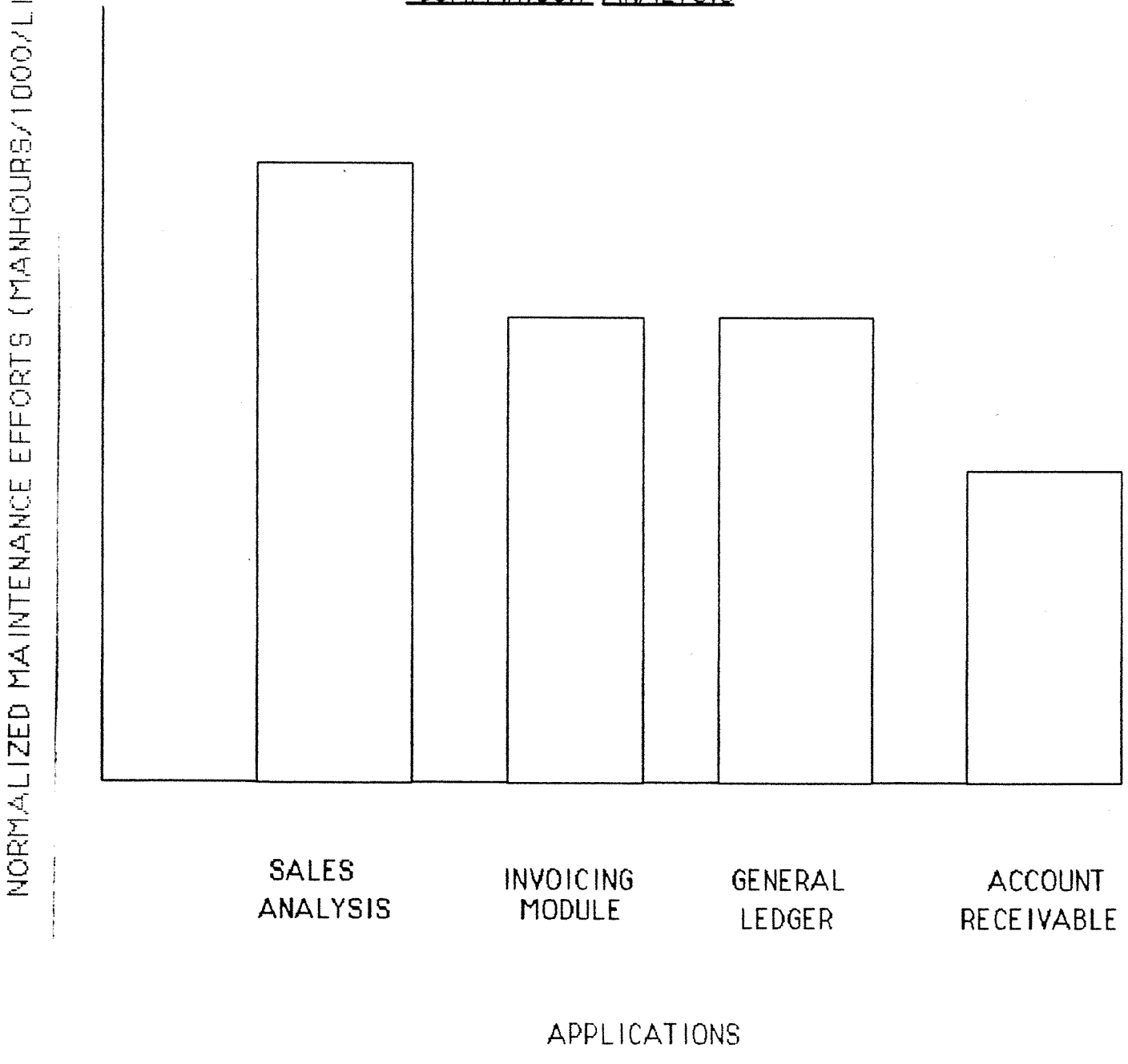
FINANCIAL ANALYSIS				
MAINTENANCE TYPE	MANHOURS		VARIANCE	
	ACTUAL	BUGETTED	HOURS	%
1	110	200	(90)	(45)%
2	105	100	5	5%
3	95	100	5	5%

Comparison Analysis

One effective method of analysing the maintenance activities is to compare the amount of normalized effort expended in each application program in the system. Normalization can be achieved by converting the maintenance effort to a common measurement eg. manhours per 1000 code lines or manhours per function point.
An example of such analysis is shown in figure 5.3.3

FIGURE 5.3.3

COMPARISON ANALYSIS



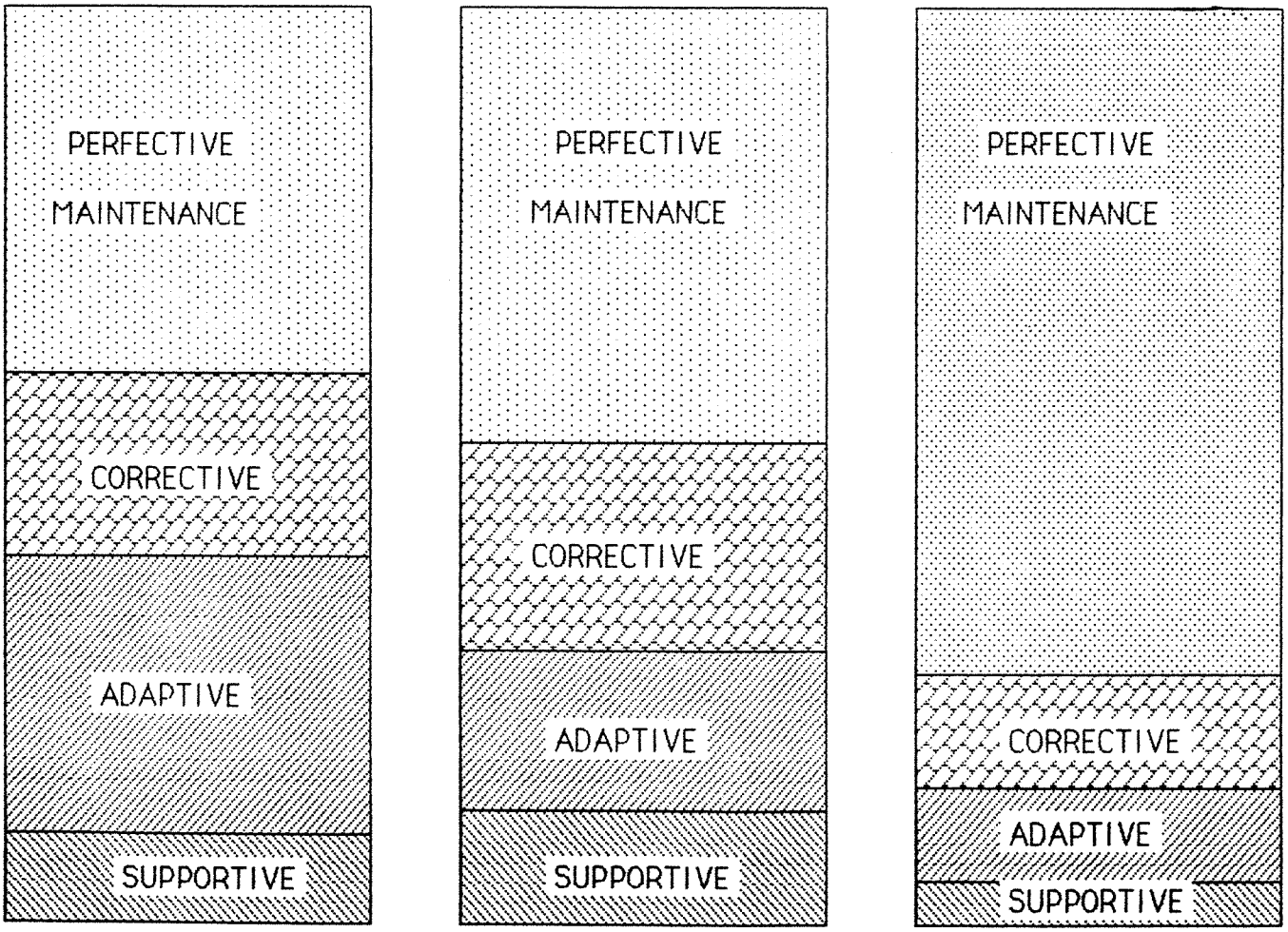
From the histogram, it can be seen that the invoicing module and the general ledger module consumed approximately the same amount of normalized maintenance resources, while sales analysis required the most normalized maintenance effort and accounts receivable required the least normalized maintenance effort.

The comparison analysis between the "normal" effort application programs and the "abnormal" effort application programs (on the aspects of program characteristics and organisational maintenance tasks) can yield valuable insights into the potential strengths and weaknesses of the maintenance framework.

Resources Allocation Analysis

Few DP organizations have statistics on the deployment of personnel classified according to maintenance type(7). A sample of such an analysis is shown in figure 5.3.4

FIGURE 5.3.4



RESOURCES ALLOCATION ANALYISIS

The advantage of this analysis is that the DP manager can identify the areas where time and effort could be devoted to improving the maintenance function.

In the above example, 80% of the maintenance effort is expended on the perfective maintenance. Therefore to improve the overall maintenance function, the DP manager should consider investing on productivity tools (eg. dynamic analyzer) that improve the perfective maintenance efficiency.

Metric Analysis

A series of measurements can be utilized to assist a DP manager to analyze the quality of the maintenance effort. These types of analysis use metrics, which are measurements showing the relationship between two variables. For example,the relationship between number of control structures and maintenance effort. The use of metrics can involve statistical tools such as regression analysis and linear programming(8). The computed metrics are compared with a standard to determine the quality of the maintenance effort. Section 5.4 of this report includes the detailed use of metrics in evaluating the effectiveness of the maintenance function.

A common feature of the above analysis is that it depicts a form of management by exception (Drucker 1979). Management attention is focused on issues (in this case application programs) that deviate from the expected standard. The primary benefit of such a philosophy is that it can be used to concentrate the use of scarce resources on the most needed area.

REFERENCES

1. Perry W. E.
"Managing Systems Maintenance"
QED Information Science Inc. Wellesley , Massachusetts 1981, pg 31-53

2. Perry W. E. - Op Cit pg 32

3. Perry W. E. - Op Cit pg 33

4. Perry W. E. - Op Cit pg 35

5. Perry W. E. - Op Cit pg 35

6. Perry W. E. - Op Cit pg 44

7. Perry W. E. - Op Cit pg 48

8. Riggs R.
"Computer Systems Maintenance"
Datamation, Nov 1969, pg 227-232

5.4 SOFTWARE MAINTENANCE MEASUREMENT METHODOLOGY

5.4.0 INTRODUCTION

For effective maintenance management decisions, DP management must be able to determine the effectiveness of the maintenance process. Measures are needed to differentiate good and effective maintenance from poor or ineffective maintenance.

These measures help DP management to make the following maintenance decisions(1) :

1. Whether additional resources should be spent on a particular application program or whether a complete rewrite of the program is needed.

2. How effective a particular maintenance technique is?

3. How efficient a particular maintenance technique is?

5.4.1 DEFINING MAINTAINABILITY

The failure of DP managers to understand the concepts of software maintainability, and the inability to measure this concept has contributed to the in high cost of software maintenance (2). Moreover, this inability has often resulted in the DP manager misapplying substantial amounts of resources in maintenance tasks.

Boehm(3) suggests that for software to be maintainable, it must possess 3 qualities; testability, understandability and modifability. He defines "testability" as the ease of demonstrating the correctness of the software maintenance, "understandability" as the extent to which one can read and understand the software code and documentations and "Modifiability" as the ease with which one can modify the software code.

Martin(4) expands Boehm's definition of maintainability to reflect the different types of maintenance. Martin advocates that a high degree of maintainability should imply a high degree of testability, understandability, modifiability, portability, reliability, usability and efficiency.

5.4.2 MAINTENANCE MEASUREMENT METHODS

Having broken down software maintainability into its basic components, the next step is to develop criteria for its measurement. Basically there are 2 approaches to maintenance measurement :

1. subjective judgement approach

2. maintenance metrics approach

Subjective Judgement Approach

The subjective judgement method relies heavily on individual experience, knowledge and intuition in measuring software maintainability. The attractive feature of this measuring methodology is that it is easly to gather the necessary data and at a relatively lower cost compared with the program metrics method.

On the other hand, this method suffers a number of deficiencies(5) :

1. judgment can be difficult to substantiate

2. it may be biased

3. it tend to be unduly influenced by present needs and tends to ignore future requirements

4. judgements may be inconsistent between time, place and different people

These deficiencies, however, do not necessarily render the subjective judgement method useless. This is due to the fact that not all attributes of maintainability are quantifiable. For example user's satisfaction, programmer morale and complexity of the maintenance environment cannot be measured in quantifable terms(6). To capture these attitudinal facets of maintainability we need judgemental methodology such as the Likert scale. The Likert scale measuring method involves asking the person concerned to judge a specific measurement criterion and to place it somewhere between two opposite end of a scale. A typical Likert scale is illustrated in figure below :

Example of Likert Scale

Please tick the appropriate box.

Morale of maintenance programmer

1. Low
2. Rather low
3. Moderate
4. Rather high
5. High

Maintenance Metrics

The other method of measuring maintainability is the maintenance metrics approach.

Maintenance metrics attempt to measure maintainability by measuring the strength of the relationship between two correlated variables that influence software maintainability. For example, maintenance expenditure and frequency of maintenance.

The reliability of these metrics depends on the degree of correlation between the influencing variables.

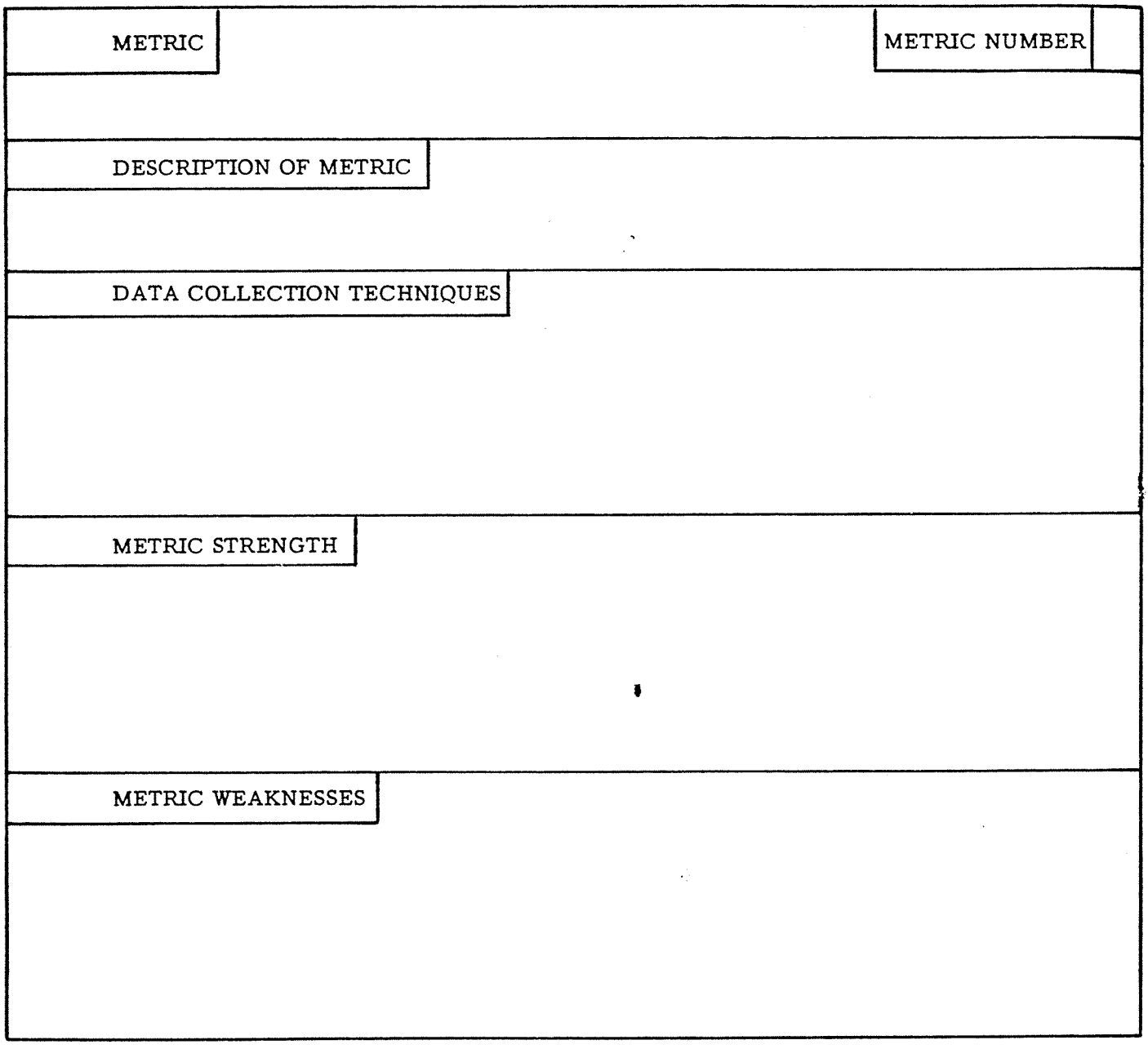
The more significant the relationship, the more reliable is the metric in measuring maintainability.

Maintenance Metrics Reports

Maintenance metrics are usually recorded in a report form. An example of such a report is shown in figure 5.4.1

Figure 5.4.1

MAINTENANCE METRICS DESCRIPTION



Adapted from Perry W.E. - Op Cit pg 294

The key features of this metrics report include (7) :

1. Metric Name

Usually this consists of keywords of the variables included in the metric eg expenditure/frequency metric.

2. Metric Description

A detailed description of the meaning of the variables comprising the metric.

3. Metric Measurement

This includes tools and techniques used to measure the variables which constitute the metric.

4. Metric Strength and Weakness

Metric strengths are the aspects of maintenance where the metric is particularly valuable as good indicator. Metric weaknesses are aspects of software maintainability which may be misrepresented by the metric.

Advantages of Maintenance Metrics

Many advantages accrue from the maintenance metrics measurement approach.

These include :

1. the measurement is objective

2. the measurement is backed up by available literature

3. the measurement is consistent regardless of time, place or people

5.4.3 INTEGRATED MEASUREMENT METHODOLOGY

A more effective software maintenance measurement methodology involves the combination of the judgement and maintenance metrics approaches.

Under the integrated approach, judgemental criteria are used to measure the non-quantifiable aspects of maintainability while the quantifiable aspects are measured by program metrics.
The main advantage of this approach is that it provides a comprehensive and realistic indicator to software maintainability. However, such an approach may be expensive especially for the small company.

The rest of this chapter focuses attention on how this integrated

approach can be used to operationalize software maintainability (ie program understandability, testability, modifiability, portability, reliability, usability and efficiency).

5.4.4 UNDERSTANDABILITY

Software understandability can be defined as the ease with which a person (with reasonable programming skill) is able to understand the function of the program by reading its source code and associated documentation. Software understandility can be characterised by the following properties(8) :

Program structure
Program consistency
Program conciseness
Program clarity
Program completeness
Program complexity

Program Structure

A well-structured program is one which adheres to a set of general structuring rules. (eg Jackson structure programming methodology (9)).

Program structure can be measured by using a checklist (a judgement measuring approach) such as that shown in figure 5.4.2 or by using automatic structure checker program metric tools(10).

Program Consistency

A program is considered consistent if it adheres to a consistent coding style and design approach(11). For example the use of table-driven menus throughout the program. Program consistency can be assessed by using the checklist shown in figure 5.4.2

Program Conciseness

A concise program is one which has no excessive statements or procedures(12). Automatic flowcharting programs(13) and a checklist (figure 5.4.2) can be used to assess this feature by identifying the presence of any "dead code" in the program.

Program Completeness

A program is considered complete if all its components are present and fully developed(14).

Program completness can be assessed by using checklist shown in figure 5.4.2

Program Complexity

The more complex the program, the more difficult will be the understanding of the program.

Program complexity can be measured by 4 main metrics(15) :

1. module size

2. Halstead's software science metrics eg. size

3. McCabe's cyclomatic number

4. McClure's control variable complexity

Module Size

The module size metric assumes that the complexity of the program is directly related to the size of the module(16).

IBM(17) in their research concluded that individual module should not exceed 50 lines of code or should not exceed a single page in the source listing.

Note, however this metric is only a rough indication of program complexity as there are other important variables (eg program control structure) that influence program complexity.

Software Science Metrics

Halstead's software science metrics(18) are built up from comprises 4 basic elements :

1. the number of distinct operators in program (n1)

2. number of distinct operands in program (n2)

3. total number of operators in program (N1)

4. total number of operands in program (N2)

The operators include both arithmetic and logic operators while the operands include variables and constants.

Halstead states that the program size (N) is equal to the summation of (N1) and (N2) . According to him, the larger the value of N1, the more difficult it would be to understand the program.

Elshoff(19) devises an estimation formula to measure the value "N"

Elshoff's metric can be represented as

^N = n1log2n1 + n2log2n2

where ^N is the estimated length of the program and n1 and n2 are defined above.

McCabe's Cyclomatic Number

McCabe's metric (20) assumes that program complexity is influenced by the number of linearly independent paths through a program known as cyclomatic number. The cyclomatic number can be determined by the following equation

$$N = C + 1$$

where N is the cyclomatic number and C is the number of comparisons in the program.

McCabe recommends that for any structured design program, the value of "C" should not exceed 10 for any module. An example of an automated tool that can be used to calculate cyclomatic number is "Flow"(21).

McClure's Control Variable Complexity

McClure(22) suggests that program complexity is influenced by 2 variables

1. the number of comparisons in the module

2. the number of control variables used in the module

McClure's complexity metric can be summarised in the following equation

$$N = C + V$$

where "N" is the complexity of the module, "C" is the number of comparisons in the module and "V" is the number of control variables referred in the module.

5.4.5 RELIABILITY

Reliability can be defined as the extent to which the software correctly performs its functional specifications (23). A reliable program has to be correct, complete and consistent. Software reliability can be measured by using a checklist (see figure 5.4.2), automated measurement tools such as execution path tracers(24) and program metrics.

The program metrics include :

1. Gilb's bebugging metric

2. Shneiderman's probability metric

3. error statistics metric

Gilb's Bebugging Metric

Gilb's bebugging metric(25) assumes that the reliability of the program is directly related to the number of errors removed from the program. Basically, Gilb's bebugging metric involves 3 steps.

1. erroneous data (seeded errors) are intentionally inserted into the test data

2. the program is then tested for both seeded and real errors

3. the number of seeded and real errors detected are input into a formula

$$N = (R \times T) / S$$

where N is a measurement of reliability ,"R" is the number of real errors found, "S" is the number of seeded errors found and "T" is the total number of seeded errors planted in the test data. According to Gilb, the lower the value "N", the more reliable the program.

Shneiderman's Probability Metric

Shneiderman(26) defines software reliability as the probability that a particular application program will operate for certain period of time without software errors.

Shneiderman probability metric is based on the mean time between two errors (MTBE). The higher the MTBE the more reliable the program.

Error Statistics Metric

The basic assumption underlying this metric is that the more errors found in a particular module during the program testing phase the more likelihood that such module will require maintenance in future(27).

Note however, program complexity too has impact on the reliability of a program. This is because the more complex the program, the more error-prone it is likely to be (28). The complexity measures discussed earlier are also applicable in

measuring program reliability.

5.4.6 TESTABILITY

Testability is the ease with which the correctness of software maintenance can be demonstrated(29).

The simpler the program, the easier it can be tested for changes. A program is considered simple if it is modular, well-designed and well-structured, with minimum complexity(30).

Testability can be estimated by using a checklist (see figure 5.4.2). Maintenance metrics used to measure program complexity and automated tools such as automatic complexity analysers(31).

5.4.7 MODIFIABILITY

Program modifiability is the ease with which the software design/code can be changed. Program modifiability can be characterised as(32) :

1. understandability

2. generality

3. flexibility

4. simplicity

These facets of program modifiability can be estimated by using a checklist (see figure 5.4.2) and the following maintenance metrics.

1. change exercise metric

2. Stevens' criteria

Change Exercise Metric

Mathematically, the change exercise metric(33) can be represented as

$$N = A / C$$

where "N" is the degree of modifiability "C" is the average complexity value for modules in the application system and "A" is the average complexity of the particular modules to be changed.

The larger the value "N" the more difficult to change the module.

Note that the complexity variable can be measured by using

complexity metrics mentioned earlier.

Stevens' Criteria

Steven(34) suggests that program modifiability is influenced by 2 variables, program cohesiveness and program coupling.

Program cohesiveness is defined as the individual module internal strength (ie the strength of the relationship of its internal elements).

Program coupling is defined as the degree of interdependence among modules.

Steven advocates that a modifiable program is one with low coupling and high cohesiveness.

5.4.8 PORTABILITY

Program portability refers to the extent to which the software can be operated on different computer configurations without major modification. Major characteristics of program portability include (35) :

1. program structure

2. program flexibility

3. program dependence on particular hardware or operating system

Program portability can be assessed by using the checklist shown in figure 5.4.2 and software quality measurement tools such as standard language version (36).

5.4.9 PROGRAM EFFICIENCY

Program efficiency is defined as the extent to which a program economically uses the computer resources to perform its intended function(37). Program efficiency can be assessed by using a checklist developed by Van Tassel(38) (see figure 5.4.2) and automated measurement tools such as a structure checker and performance monitors (39).

5.4.10 USABILITY

A usable program is one that is easy and convenient to use. The major attributes of program usability include (40) :

1. reliability

2. portability

3. efficiency

4. user friendliness

These attributes can be assessed by using the checklist shown in figure 5.4.2 and program metrics discussed earlier.

This chapter presents a comprehensive maintenance measurement methodology. It is not intended that the DP department should adopt all measurement methods discussed. However, for effective maintenance decisions to be made , a representative measure of each facet of software maintainability should be adopted.

FIGURE 5.4.2

CHECKLISTS

Understandability checklist*

Structuredness

1. Is the program modularized and well-structured? (See Chapter 4 for a list of general structuring rules.)

Documentation

2. Is the program documented? Minimal documentation for a well-structured program requires a comment block for each module, subroutine, or subprogram that explains:
   (a) What the module does in one or two brief sentences.
   (b) A list of the program variables whose values may be modified in this module.
   (c) A list of the modules that invoke this module.
   (d) A list of the modules that this module invokes. (See Chapters 8 and and 9 for a detailed discusssion of documentation.)
3. Is other useful commentary material included in the program? This would include:
   (a) Inputs and outputs
   (b) Accuracy checks
   (c) Limitations and restrictions
   (d) Assumptions
   (e) Error recovery procedures for all foreseeable error exits
   (f) Modification history
   (g) Date written and date last changed

Consistency

4. Is a consistent indentation and spacing style used throughout the program?
5. Is there at most one executable statement per line of code?
6. Are all variable names and procedure names unique, descriptive, and in compliance to company standards?
7. Does each variable and each procedure have one and only one unique name in the program?
8. Is each variable used to represent one and only one quantity, and each procedure used to represent one and only one logical function?
9. Is the program a true representation of the design; that is, is the integrity of the design preserved throughout the entire program?
   (a) Does the program neither add to nor subtract from the design algorithm?
   (b) Is the design structure exactly and explicitly represented in the code?

(Continued)

(Continued)

10. Are all elements of an array/table functionally related?
11. Are parentheses used to clarify the evaluation order of complex arithmetic and logical expressions?

Completeness

12. Are cross-reference listings of variable names and a map of calling and called subroutines supplied?
13. Are all external references resolvable and all input/output descriptions available?
14. Does the program contain all referenced subprograms not available in the usual system library?
15. Are all unusual termination conditions described?
16. Are error recovery procedures included?
17. Are error messages descriptive and clearly displayed?

Conciseness

18. Is all code reachable?
19. Are all variables necessary?
20. Is redundant code avoided by creating common modules/subroutines?
21. Is there a transfer to all labels?
22. Is division of the program into an excessive number of modules, overlays, functions, or subroutines avoided?
23. Are expressions factored to avoid unnecessary repetition of common subexpressions?
24. Does the program avoid performing complementary operations on the same variable(s) such that removal of these operations leaves the program unchanged?
25. Does the program avoid poorly understood and nonstandard language features?

*Many of the questions have been adapted from Boehm's [1] checklists.

Reliability checklist

1. Does the program contain checks for potentially undefined arithmetic operations (e.g., division by zero)?
2. Are loop termination and multiple transfer index parameter ranges tested before they are used?
3. Are subscript ranges tested before they are used?
4. Are error recovery and restart procedures included?
5. Are numerical methods sufficiently accurate?
6. Are input data validated?
7. Are test results satisfactory (i.e., do actual output results correspond exactly to expected results)?
8. Do tests show that most execution paths have been exercised during testing?
9. Do tests concentrate on most complex modules and most complex module interfaces?
10. Do tests cover the normal, extreme, and exceptional processing cases?
11. Was the program tested with real as well as contrived data?
12. Does the program make use of standard library routines rather than develop its own code to perform commonly used functions?

1. Is the program modularized and well-structured?
2. Is the program understandable?
3. Is the program reliable?
4. Can the program display optional intermediate results?
5. Is program output identified in a clear, descriptive manner?
6. Can the program display all inputs upon request?
7. Does the program contain a capability for tracing and displaying logical flow of control?
8. Does the program contain a checkpoint-restart capability?
9. Does the program provide for display of descriptive error messages?

Efficiency checklist*

1. Is the program modularized and well-structured?
2. Does that program have a high degree of locality—that is, the program uses only a small subset of its pages at any point during execution—to aid efficient use of virtual memory?
3. Are unused labels and expressions eliminated to take full advantage of compiler optimization?
4. Are exception routines and error-handling routines isolated in separate modules?
5. Was the program compiled with the use of an optimizing compiler?
6. Was as much initialization (e.g., initializing arrays, variables, storage allocations) as possible done at compilation time?
7. Is all invariant code—that is, code which does not need to be processed within a loop—processed outside the loop?
8. Are fast mathematical operations substituted for slower ones? (For example, I + I is faster than 2*I.)
9. Is integer arithmetic instead of floating-point arithmetic used when possible?
10. Are mixed data types in arithmetic or logical operations avoided when possible to eliminate unnecessary conversions?
11. Are decimal points of operands used in arithmetic aligned when possible?
12. Are program variables aligned in storage?
13. Does the program avoid nonstandard subroutine or function calls?
14. In an *n*-way branch construct, is the most likely condition to be TRUE tested first?
15. In a complex logical condition, is the most likely TRUE expression tested first?
16. Is the most efficient data type used for subscripts?
17. Are input/output files blocked efficiently?

*Many of the questions are adapted from Van Tassel [17].

Portability checklist

1. Is the program written in high-level, machine-independent language?
2. Is the program written in a widely used standardized programming language, and does the program use only a standard version and features of that language?
3. Does the program use only standard, universally available library functions and subroutines?
4. Does the program use operating system functions minimally or not at all?
5. Are program computations independent of word size for achievement of required precision or memory-size restrictions?
6. Does the program initialize memory prior to execution?
7. Does the program position input/output devices prior to execution?
8. Does the program isolate and document machine-dependent statements?
9. Is the program structured to allow phased (overlay) operation on a smaller computer?
10. Has dependency on internal bit representation of alphanumeric or special characters been avoided or documented in the program?

Usability checklist*

1. Is the program self-descriptive from the user perspective?
   (a) Are explanations of how the program works and what the program does available in different levels of detail with examples included?
   (b) Is a HELP feature pertinent to any dialogue situation included?
   (c) Is a correct, complete explanation of each command and/or operating mode available on request?
   (d) Can the user become thoroughly acquainted with the program usage without human assistance?
   (e) Is current program status information readily available on request?
2. Does the program provide the user with a satisfying and appropriate degree of control over processing?
   (a) Does the program admit interruptions of a task to start or resume another task when operating in interactive mode?
   (b) Does the program admit process canceling without detrimental or unexpected side effects?
   (c) Does the program allow the user to make background processes visible?
   (d) Does the program have a command language that is easy to understand and allows clustering of commands to build "macros"?
   (e) Does the program provide detailed prompting when requested to help the user find his way through the system?
   (f) Does the program provide understandable, nonthreatening error messages?

3. Is the program easy to learn to use?
   (a) Is the program usable without special DP knowledge?
   (b) Are input formats, requirements, and restrictions completely and clearly explained?
   (c) Is user input supported by a menu technique in interactive systems?
   (d) Does the program offer error messages with correction hints?
   (e) For interactive systems, are manuals "on-line"? For batch systems, are manuals readily available?
   (f) Are manuals written using user terminology?
4. Does the program make use of a data management system to automatically perform clerical/housekeeping activities and manage formatting, addressing, and memory organization?
5. Does the program behave consistently in a manner that corresponds to user expectations?
   (a) Does the program have a syntactically homogeneous command language and error message format?
   (b) Does the program behave similarly in similar situations by minimizing variances in response times?
6. Is the program fault-tolerant?
   (a) Can the program tolerate typical typing errors?
   (b) Can the program accept reduced input when actions are to be repeated?
   (c) Can commands be abbreviated?
   (d) Does the program validate input data?
7. Is the program flexible?
   (a) Does the program allow for free-form input?
   (b) Does the program provide for repeated use without the need for redundant specification of input values?
   (c) Are a variety of output options available to the user?
   (d) Does the program provide for omission of unnecessary inputs, computations, and output for optional modes of operation?
   (e) Does the program allow the user to extend the command language?
   (f) Is the program portable?
   (g) Does the program allow the user to define his own set of functions and features?
   (h) Can the program be seen in a subset mode?
   (i) Does the program allow the experienced user to work with a faster version, allowing abbreviated commands, default values, and so on, and inexperienced users to work with a slower version, providing a help command, monitoring capabilities, and so on?

Adapted from Martin J. and McClure C. - Op Cit pg 43 - 57

REFERENCES

1. Perry W.E.
   "Managing Systems Maintenance"
   Q.E.D. Information Sciences Inc 1981 pg 287 - 288

2. Martin J. and McClure C.
   "Software Maintenance : The Problem and Its Solutions"
   Prentice-Hall, Inc, NJ 1983 pg 43

3. Boehm B. and Others
   "Characteristics of Software Quality"
   New York : TRW/North-Holland Publishing Co 1978 pg 3.1 - 3.26

4. Martin J. and McClure C. - Op Cit pg 44 - 46

5. Perry W.E. - Op Cit pg 288 - 289

6. Perry W.E. - Op Cit pg 288 - 289

7. Perry W.E. - Op Cit pg 292 - 293

8. Boehm J and Others - Op Cit pg 3.1 to 3.26

9. Jackson M.A.
   "Principles of Program Design"
   Academic Press ,1975

10. Martin J. and McClure C. - Op cit pg 404 - 412

11. Boehm B. and Others. - Op Cit Pg 3.1 to 3.26

12. Boehm B. and Others. - Op Cit Pg 3.1 to 3.26

13. Martin J. and McClure C. - Op Cit pg 404 - 412

14. Boehm B. and Others. - Op Cit Pg 3.1 to 3.26

15. Martin J. and McClure C. - Op Cit Pg 52 - 59

16. Martin J. and McClure C. - Op Cit Pg 52

17. Walston L.E. and Felix C.P.
"A Method of Program Measurement and Estimation"
IBM Systems Journal Vol 16, No 1, 1977 pg 54 - 73

18. Halstead H.M.
"Elements of Software Science"
North-Holland Elsevier, New York 1977 pg 84 - 91

19. Elshoff J.
"Measuring Commercial PL/1 Programs Using Halstead's Criteria"
ACM SIGPLAN Notices, May 1976 pg 38 - 46

20. McCabe T.
"A Complexity Measure"
IEEE Trans on Software Engineering Vol SE-2 No 4, Dec 1976 pg 308 - 320

21. Martin J. and McClure C. - Op Cit pg 57

22. McClure C.
"Reducing COBOL Complexity Through Structured Programming"
Van Nostrand Reinhold Co Ny, 1978 pg 77 - 121

23. Glass R.L. and Noiseux R.A.
"Software Maintenace Guidebook"
Prentice-Hall Inc NJ 1981 pg 33

24. Martin J. and McClure C. - Op Cit pg 404 - 412

25. Gilb T.
"Software Metrics"
Winthrop Publishers Inc Cambridge MA 1977 pg 26 - 49

26. Shneiderman B.
"Software Psychology : Human Factors in Computer and Information System"
Winthrop Publishers Inc Cambridge MA 1980 pg 93 - 122

27. Martin J. and McClure C. - Op Cit pg 62

28. Martin J. and McClure C. - Op cit pg 62

29. Martin J. and McClure C. - Op Cit pg 63

30. Martin J. and McClure C. - Op Cit pg 63

31. Martin J. and McClure C. - Op Cit pg 404 - 412

32. Martin J. and McClure C. - Op Cit pg 64

33. Martin J. and McClure C. - Op Cit pg 66

34. Stevens W. and Others
"Structured Design"
IBM Systems Journal Vol 13 No 2 1974 pg 115 - 139

35. Martin J. and McClure C. - Op Cit pg 67

36. Martin J. and McClure C. - Op Cit pg 404 - 412

37. Glass R.L. and Noiseux R.A. - Op Cit pg 34 - 35

38. Tassel D.V.
"Program Style, Design, Efficiency, Debugging and Testing"
Prentice-Hall Inc NJ 1978 pg 238 - 284

39. Martin J. and McClure C. - Op Cit pg 404 - 412

40. Martin J. and McClure C. - Op Cit pg 71

5.5 SOFTWARE MAINTENANCE METHODS

An ideal piece of software is one which requires no future maintenance. To be able to achieve this, the software developer must incorporate in the software all anticipated needs of the end-users. This is not only impractical but also impossible.

Hence, the next closest thing to "perfect" software is software which lessens future modification and, if maintenace is required, minimum effort is needed to implement it. The activities needed to achieve this aim are known as "preventive maintenance " The key difference between preventive maintenance and normal maintenance, known as corrective maintenance, is that preventive maintenance is performed prior to any software failure. Preventive maintenance is an up-front activity performed before the maintenance cycle begins. In fact, preventive maintenance starts at the specification phase of the software development cycle and continues through the maintenance phase.

Corrective maintenance on the other hand is initiated only with the intention of restoring the application software to an acceptable standard and commences only when the software is put to operation.

ADVANTAGES OF PREVENTIVE MAINTENANCE

There are many advantages of the preventive maintenance concept. These include :

1. Minimize Software Costs

One of the objective of preventive maintenanace is to ensure any potential foreseeable bug or misspecification of need does not go undetected in the early stages of software development. Figure 5.5.1 clearly demonstrate that the cost of correcting a software bug or misspecification of need grows expotentially with each stage of the software life cycle. Examples of preventive measures that aim to achieve this objective are a quality assurance program and rapid prototying.

2. Facilitate Maintenace Operation

As preventice maintenance is a planned activity, it can be scheduled at a time that causes least inconvenience to the operational environment. Moreover, the maintenance activities can be planned to suit the availability of existing resources (machine time and DP personnel). Though attractive in theory, it is doubtful that these advantages can be achieved in practice.

3. Less Disruptive Emergency Maintenance

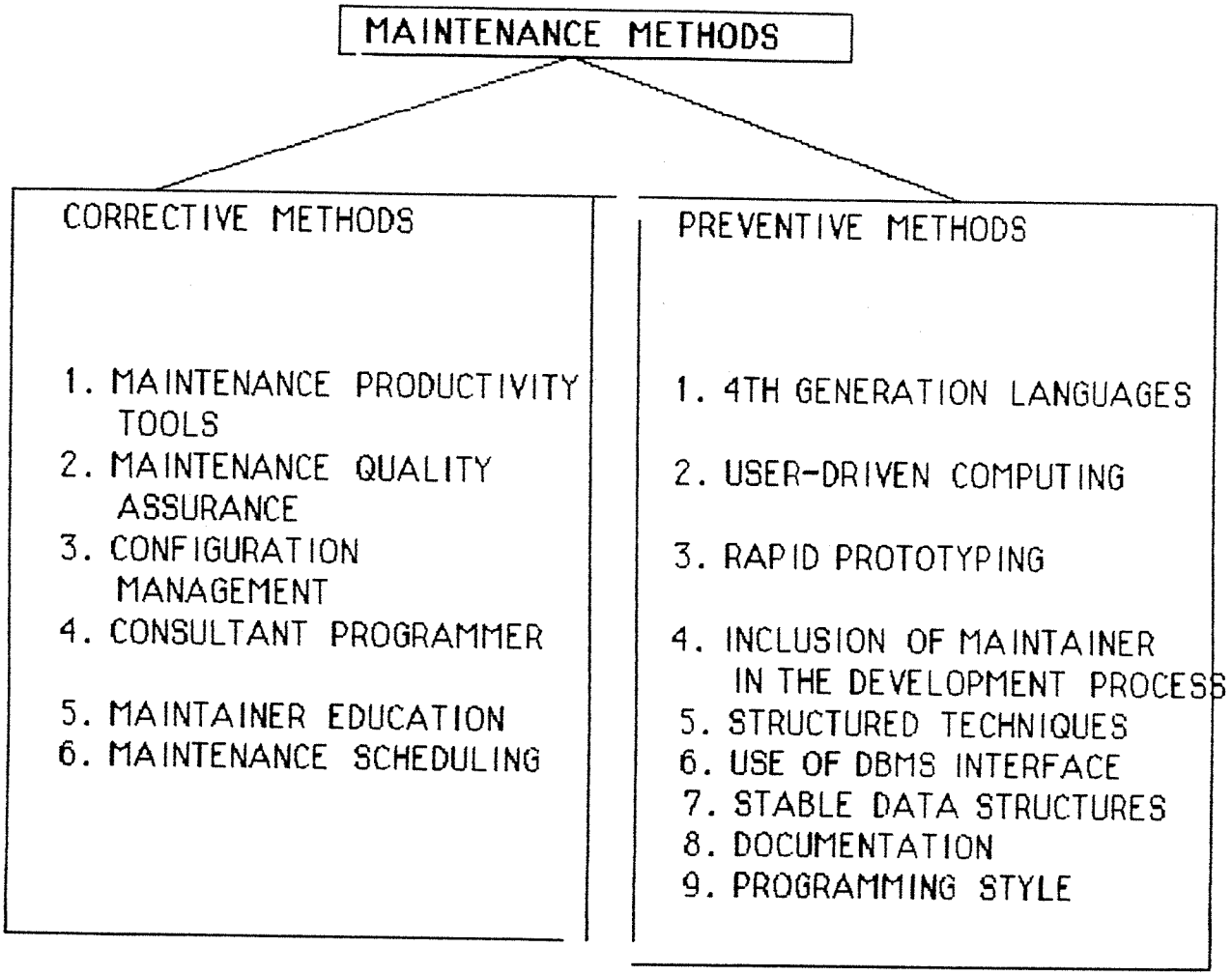
Proper preventive maintenance will result in lowering the incidence of disruptive emergency maintenance.

4. Facilitate Maintenance Planning

Preventive maintenance enables the DP management to plan and schedule staff requirements with a greater degree of accuracy. This information facilitate planning for staff loading when the maintenance curve is at its peak and staff unloading when the maintenance curve is at its trough.

A summary of preventive maintenance methods and corrective maintenance methods is shown in figure 5.5.1

FIGURE 5.5.1


MAINTENANCE METHODS
CORRECTIVE METHODS
1. MAINTENANCE PRODUCTIVITY TOOLS
2. MAINTENANCE QUALITY ASSURANCE
3. CONFIGURATION MANAGEMENT
4. CONSULTANT PROGRAMMER
5. MAINTAINER EDUCATION
6. MAINTENANCE SCHEDULING
PREVENTIVE METHODS
1. 4TH GENERATION LANGUAGES
2. USER-DRIVEN COMPUTING
3. RAPID PROTOTYPING
4. INCLUSION OF MAINTAINER IN THE DEVELOPMENT PROCESS
5. STRUCTURED TECHNIQUES
6. USE OF DBMS INTERFACE
7. STABLE DATA STRUCTURES
8. DOCUMENTATION
9. PROGRAMMING STYLE

5.5.1 CORRECTIVE MAINTENANCE METHODS

5.5.1.1 MAINTAINER PRODUCTIVITY TOOLS

In the past, dumps and traces were the only maintenance tools. However, with the ever increasing cost of maintenence, we now need more efficient and productive tools to deal with the maintenance process.

Good tools are vital to the maintenance task. Their proper usage can enhance program quality and increase the software maintainer's productivity. The importance of good maintenance tools is illustrated by the study done by Van Tassel(1). Van Tassel's research revealed that a good debugging compiler can result in doubling the productivity level of the software maintenance process.

TYPES OF MAINTAINER PRODUCTIVITY TOOLS

Basically, maintainer productivity tools can be divided into 3 major classes :

1. Testing and Debugging Tools

These tools are related to the testing and debugging activities of the maintenance function. Tools included in this category are :

- language processor
- operating system
- debugging tools
- comparator
- dynamic and static analyzer
- code optimizer
- verification tools
- dumps
- cross-reference listing
- data dictionaries
- test data generators
- language conversion packages
- data management system
- structuring engines
- source code reformatter

2. Documentation Tools

These tools deal with the documentation activities in the maintenance process. Important tools in this category are :

- text editor
- reformatters

- cross-reference listing
- source management libraries
- data flow analyzer
- automatic documentors
- automatic flowchart and structure diagrams
- module invocation analyzer
- structuring engines

3. Design Tools

The prime objectives of these tools are to encourage user participation in the design phase during software maintenenace and to produce a design which require less maintenance effort in future(2).

Tools belonging to this category include :

- scheduling guidelines
- prototyping
- data flow diagrams
- data dictionaries
- logic flowcharts
- program abstracts
- program design walkthroughs
- structural design tools.

TESTING AND DEBUGGING TOOLS

1. Language Processor (compiler)(3)

The language processor or compiler is the most important piece of software tool in the maintenance process.

The output generated by the compiler should include :

a. Procedure oriented cross-reference listing

   This listing shows the presence of cross-reference between the variables and the precedence sets in which the variables are used. This listing is of particular importance in situations where sharing of a common set of data resources exist.

b. Data structure reference listing

   A data structure reference listing attempts to show not only the names of all data items but also its constituent elements.

The reference listing permits the maintainer to have an idea of the environment in which each data field operates.

2. Link Editor

The main function of the link editor is to provide access at the intra-program level and at the same time satisfying the external definition. This is particularly useful to a maintainer wishing to test a program module without worrying about the external variable contraints.

3. Operating System

An important feature of the operating system is the presence of build-in debugging tools.

A debugger permits easy location and correction of program errors.

An example of a debugging tool is the symbolic dump program(4). This utility not only generates information about global variables but also presents information on local variable values in all activated procedures.

In addition, it permits the maintainer to specify conditions relating to the object values and all name value pairs satisfying these conditions.

4. Comparator

A comparator is a program utility which is capable of analyzing a set of information (eg. program) and output any difference detected between two programs in a useful form. For example, this utility permits the maintainer to compare two different versions of a same program and highlight the differences in their performance(6).

5. Static Analyzer

A static analyzer is useful to a maintainer in three ways (7):

a. it enables the programmer to unravel the program logic in a graphic form

b. it provides information regarding program style and complexity

c. the static analyzer utility does not require the program to be executed but instead it scans through the text and searches for anomalies which are likely to result in program errors

This tool is particularly useful in an environment where testing a modified program could endanger existing live data.

TEST DATA GENERATORS

Test data generators are programs which are used to generate test input data to check the reliability of the modified program. This is an important maintainer tool especially if the modified program has to be tested in the real world environment(8).

DYNAMIC ANALYZER

A dynamic analyzer is used to detect program statements that have not been executed during a test run. This ensures that subsequent test can be designed to test these "uninvoked" statements.

The main feature of the dynamic analyzer include :

- fast syntax checking
- one step compile and run
- program path tracing
- execution suspension and restart
- variable dump and modification.

SOURCE EDITOR

Since the maintenance of the program involves changes to the source code, another important maintenance tool is a source editor. A good source editor should bespecifically oriented to the source language used.

MAINTENANCE DOCUMENTATION TOOLS

TEXT EDITOR

A substantial amount of maintenance activity is related to the alteration of the documentation to reflect the modified program. This task can be greatly reduced with a good text editor.

SPELLING CHECKER AND GRAPHICS FACILITIES

A spelling checker saves the maintainer's time of having to read through the whole documentation for spelling errors. This is particularly useful for large documents.

Graphics is another documentation productivity tool useful to maintainer. Graphics can be used for diagram production (eg. structure diagrams) and lay out complicated tables.

REFORMATTER

The reformatter is an automatic tool capable of arranging the

documentation in a predefined format(10).

DOCUMENTATION DATABASE

A documentation database(11) is useful for organising massive documentation. It permits the maintainer to scan for texts that are associated with the current documentation. These texts can then be modified and reused. This arrangement reduces the documentation activities substantially.

MAINTENANCE DESIGN TOOLS

RAPID PROTOTYPING

Peters(12), Gladden(13), McCarden(14) and Jackson recommend the use of rapid prototyping to combat wholesale requirement changes by the end users.

Detail discussion of rapid prototyping as a maintenance design tool is presented in Section 5.5.2.3 of this report.

DATA FLOW DIAGRAMS

The data flow diagram is a visual representation of how data flows between the various functions in the system. This design tool serves as a checking device on the program design accuracy and its completeness. It can also be used to detect any unnecessary or complex procedures in the modified program.

For a detail discussion on how to use data flow diagram as a design tool see Demarco (15).

PROGRAM DESIGN TOOLS

Program design tools are used in the development of program logic. Tools that fall within this category include Nassi-Shneiderman structured flowcharts(16) and structured programming techniques.

A discussion of this technique is presented in Section 5.5.2.5 of this report.

INTEGRATED TOOLS

Each of the maintenance productivity tools mentioned so far, only focuses on one aspect of the maintenance process. Integrated tools attempt to include as many features that are helpful to the maintenance task.

An example of an integrated tool is "MAP" developed by Amdahl

Corporation. This integrated tool is capable of following functions:

1. Structure Chart View

This gives an graphic representation of the overall picture of the application program.

2. Source Code View

This permits the source code to be viewed in variety of ways

3. Data Trace

This helps to answer questions as to where the variable is defined, change or deleted.

4. Version Comparison

This permits comparison between different versions of the same program.

5. Subset Facilities

These facilities include input-output and data definition, data manipulation and control.

6. Documentation Facility

This includes source editor and reformatter.

CONCEPT OF MAINTAINER WORKBENCH

With the ever-increasing importance of software maintenance, some academics argue that special facilities should be provided to the maintainer to carry out this important task. This gives rise to the concept of maintainer workbench(17).

McClure suggested that a maintainer workbench should have the following features :

1. document production facilities

2. text editor

3. test data generator

4. static analyzer

5. configuration management of program and database facilities

6. sophisticated debugging facilities

7. integrated and regression testing facilities

REFERENCES

1. Van Tassel D.
"Program Style, Design, Efficiency, Debugging and Testing"
Prentice-Hall NJ, 1978 pg 183-187

2. Connell J. and Brice L.
"A Methodology for Minimizing Maintenance Costs"
AFIPS Conference Proceedings, May 1983 pg 113-121

3. Glass R. L. and Noiseux R. A.
"Software Maintenance Guidebook"
Prentice-Hall NJ 1981 pg 59-61

4. Sommerville I.
"Software Engineering"
International Computer Series
Addison-Wesley 1982 pg 177-178

5. Sommerville I. - Op Cit pg 180

6. Glass R.L. and Noiseux R. A. - Op Cit pg 65-67

7. Culpepper L. M.
"A System for Reliable Engineering Software"
IEEE Transactions on Software Engineering
SE1(2) 1975 pg 174-178

8. Sommerville I. - Op Cit pg 171-172

9. Sommerville I. - Op Cit pg 180

10. Glass R. L. and Noiseux R. A. - Op Cit pg 70-71

11. Sommerville I. - Op Cit pg 191

12. Peters L.
"Relating Software Requirements and Design"
ACM Proceedings of Software Quality and Assurance Workshops,
3, 1978 pg 67-71

13. Gladden G. R.
"Stop the Life Cycle, I Want to Get Off"
Software engineering Notes, 7, 1982 pg 35-39

14. McCracken D. D. and Jackson M. A.
"Life Cycle Concept Considered Harmful"
Software Engineering Notes, 7, 1982 pg 29 - 32

15. Demarco T.
"Structured Analysis and System Specification"
Yourdon Inc. NY 1978

16. Nassi I. and Scneiderman
"Flowchart Techniques for Structured Programming"
SIGPLAN Notices of ACM, 8 (1973), 8 pg 12 - 26

17. Glass R. L. and Noiseux R. A. - Op Cit pg 81

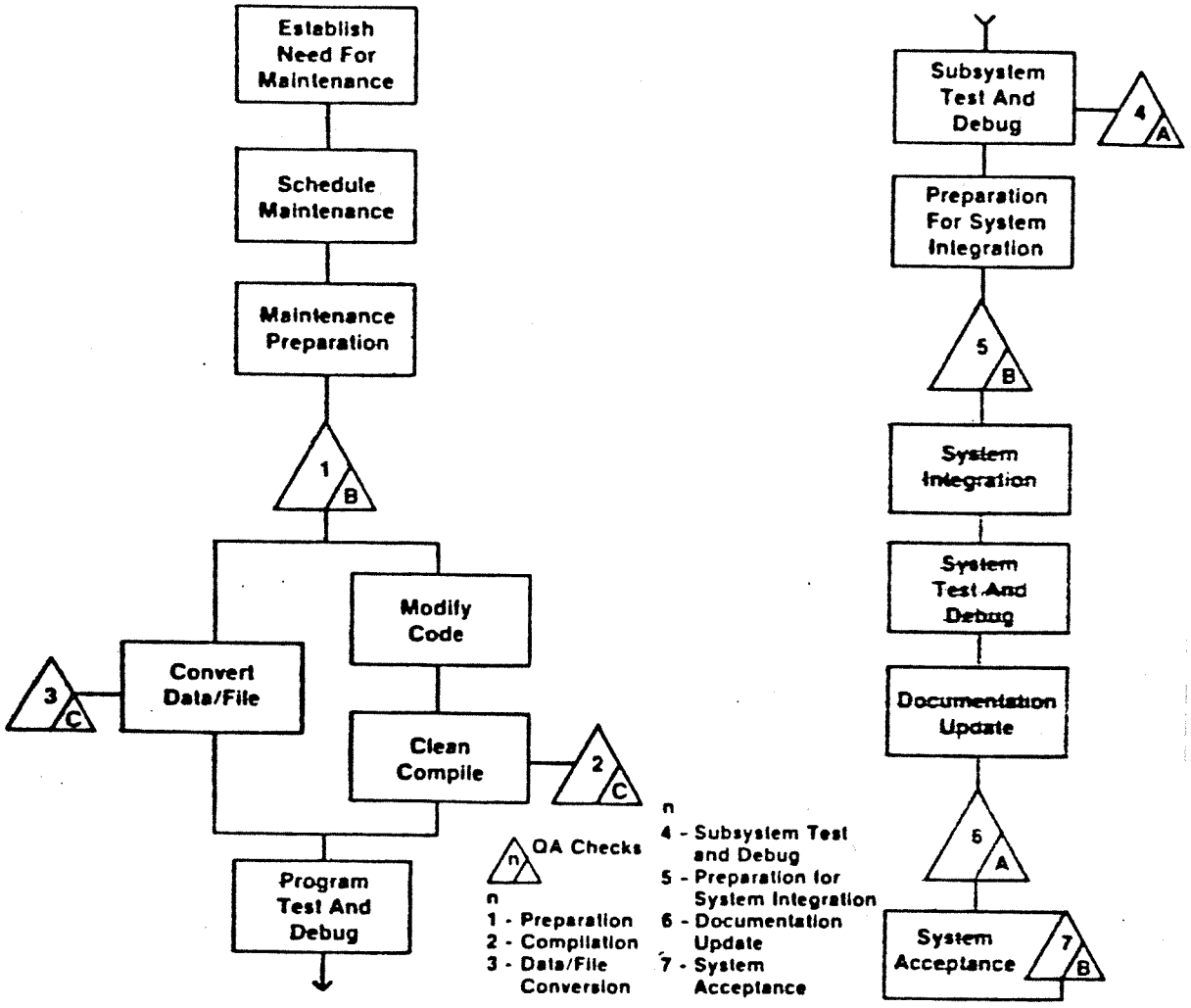
5.5.1.2 SOFTWARE MAINTENANCE QUALITY ASSURANCE

One way of reducing future corrective maintenance is through a well-defined quality assurance (QA) program . Quality can be defined in terms of adherence to standards and goals, meeting schedules and budgets, and any other criteria DP management desire to establish as a measure of quality(1).

MAINTENANCE QA PROGRAM

A Maintenance QA program is based on a set of checks or inspection points placed strategically in the maintenance process(2,3,4) Diagrammatically, such checks are at points shown in figure below

FIGURE 5.5.1.2



-Software maintenance project steps overlaid with QA checks

Adapted from Centre J. W. - Op Cit pg 405

TYPES OF QA CHECK

Basically, maintenance QA checks can be divided into 2 main classes, in-line and off-line QA checks(5).

The in-line check or gate check is mandatory and is usually performed by the a person with QA responsibility. Once the project encounters an in-line check, all maintenance activities stop till the inspection is passed by QA. Generally this check is located in the maintenance cycle where further progress with undetected errors or problems would be costly or pointless.

The offline QA check on the other hand is associated with steps that only require observation for a particular condition. For example, in a QA compilation, the only concern is to ensure that the source code is free from syntax error. Another characteristic of the off-line QA check is that if an error condition is detected, it can be quickly repaired or changed.

The off-line and in-line QA checks can be further broken down into 3 levels; A,B, and C

LEVEL OF QA CHECKS

The main criteria for distinguishing one level from another depends on the following factors (6) :

1. the detail, impact and scope of the maintenance phase

2. experience of personnel

3. degree of personnel responsibility

4. depth of personnel knowledge

5. significance of the maintenance itself

LEVEL A QA CHECK

A level A check is the most crucial and is often placed where the scope of the maintenance phase is the widest and would have serious impact on overall schedules of the maintenance process. Moreover it acts as an inspection point of the completeness and adequacy of the QA check done at levels B and C. A Level A check is usually carried out by a QA person, a separate function within the DP organization.

LEVEL B QA CHECK

Level B checks are usually delegated to the project leader performing the maintenance function. To attain objectivity, the checks should be carried out in a different physical area from

the actual maintenance . These checks have significant impact on the maintenance schedule and budget.

LEVEL C QA CHECK

This is the lowest QA check level and is usually carried out by the programmer performing the maintenance. However, a programmer undertaking the maintenance of a particular module of the program should not inspect his/her own work. Independence at level C can be achieved by having the programmer rotate or exchange his/her work with another.

The NBS special publication by Brandstad(7) provides a good tutorial in this area .

PLACEMENT OF QA CHECKS

The maintenance QA checks are represented by small triangles in figure 5.5.1.2. The 7-point checks(8) are :

1. preparatory QA

2. compilation QA

3. data/file conversion QA

4. subsystem testing and debugging QA

5. system integration QA

6. documentation update QA

7. system acceptance QA

PREPARTORY QA

The preparatory QA check falls within the in-line category and is performed at level B. The objective of this check is to ensure all necessary information needed in the maintenance process is available, and up-to-date. An oversight at this stage could seriously impinge upon subsequent maintenance phases.

This QA check should ensure that the following documents are available, referenced or otherwise accounted for(9) :

1. complete or sample data files

2. file and record descriptions

3. program descriptions, listings and specifications

4. status, background, and detailed information necessary to

execute the maintenance properly and to ensure that it is done

5. test plans and system criteria

COMPILATION QA

The primary objective of the compilation QA check is to ensure that the changes made as part of maintenance do not contaminate the original source program. The modified code is considered to have passed this inspection when no error message are generated by the compiler. This check is usually performed in an off-line mode at level C and is carried out by the programming personnel.

DATA CONVERSION QA

Data conversion QA criteria should be established prior to data conversion especially if the conversion itself is irreversible. Data conversion QA performs two functions:

1. It ensures that all directives used in the data conversion by conversion utilities or special program are free from :

   a. syntax or format errors

   b. execution errors associated with the directives

2. It ensures that all data converted are accounted for. This can be achieved by comparing the record count with the control total. Any unexpected mismatch between the two items or unexpected changes in the data size parameters such as record size and blocking factors are investigated further. This check is at level C and performed in an off-line mode.

SUBSYSTEM TESTING AND DEBUGGING QA

The objective of the subsystem testing and debugging QA check is to ensure that all the tests required by appropriate test plans are carried out . NBS Special Publication by Adrion(10) provides a brief tutorial on this testing procedure. One technique that can be used to determine the success of this check is File and Report Comparisons. This technique(11) is particularly useful for maintenance involving minor changes in the data/file structure or report contents. The criteria to be met when using this technique are :

1. no unexpected mismatch between files

2. no unexpected mismatch between reports

3. all expected differences are properly documented

Note that this is a level A check, in an off-line mode.

PREPARATION FOR SYSTEM INTEGRATION QA

The objective of preparation for integration QA check is twofold.

1. It ensures that all documentation necessary for system integration is available.

2. It ensures all previous QA checks have been performed to a satisfactory degree.

The following criteria(12) must be met at this check :

1. all structure charts must be adequately documented

2. updates to the file and record descriptions must be present or referenced

3. updates to program and routine descriptions, lists and other appropriate information must be present or referenced

4. all documented problems are cleared

This check is at level B performed in an in-line mode.

DOCUMENTATION UPDATE QA

This check is the last step to be performed before final operational testing. The objective of this check is to ensure that all documents are updated in the modified version of the program. These documents include :

1. all internal Management Information System documents for the system

2. data catalog or configuration management mechanisms

3. user documents

4. system history

This is a level A check done in an in-line gate.

SYSTEM ACCEPTANCE QA

The system acceptance QA check is the final step in the maintenance process. The criteria for this check vary with the features of the maintenance. The users of the system play an important role in determining such criteria. With well-defined and thorough QA checks imposed at earlier stages, the system acceptance QA check should not pose much problem.

PROGRAMMER QA

So far our discussion has been centered on technical QA checks of the maintenance process. There is another aspect of the QA check of the programmer QA check.

The quality of the maintenance process can be judged by evaluating the final product. This however, doesn't give any indication as to the effectiveness of the maintenance effort. The programmer QA check was a forbidden topic in the past(13), but with present escalating maintenance costs, this aspect of maintenance QA can no longer be ignored.

One technique that can be used to alleviate the programmer's fear of being audited is to use the technical peer approach (14). This technique utilizes the we-are-all-in-this together approach to eliminate or discourage those who fail to meet the group standards.

In conclusion, to achieve good software maintenance, one must not only require a good QA program but also well-qualified QA personnel to monitor the maintenance process .

REFERENCES

1. Taute B.J.
"Quality Assurance and Maintenance Application Systems"
AFIPS National Computer Conference 1983 pg 123 - 129

2. Buckley F.
"A Standard for Software Quality Assurance Plans"
Computer August 1979 pg 43 - 50

3. Dunn R. H. and Ullman R. S.
"A workable Software Quality Reliability Plan"
Proceedings of 1978 Annual Reliability and Maintainability Symposium L.A. IEEE 1978 pg 210 - 217

4. Roberts T. J.
"Maintaining Quality After The Software Is Released"
ASQC Technical Conference Transactions 31, 1977
pg 157 - 166

5. Centre J. W.
"A Quality Assurance Program For Software Maintenance"
AFIPS National Computer Conference 1982 pg 399 - 407

6. Centre J. W. - Op Cit pg 404 - 407

7. Braudstad M. and Others
"Validation ,Verification and Testing For The Individual Programmer"
NBS Special Publication 1977 pg 500 - 536

8. Centre J. W. - Op Cit pg 404 - 407

9. Centre J. W. - Op Cit pg 405

10. Adrion W. R. and Others
"Validation, Verification and Testing of Computer Software"
NBS Special Publication
U S Governmental Printing Office 1981 pg 500 - 575

11. Centre J. W. - Op Cit pg 406

12. Centre J. W. - Op Cit pg 406

13. Glass R. L. and Noiseux R. A.
"Software Maintenance Guidebook"
Prentice-Hall Inc, Englewood Cliffs N J 1981 pg 135-139

14. Glass R. L. and Noiseux R. A. - Op Cit pg 139

5.5.1.3 CONFIGURATION MANAGEMENT

The benefits of software configuration management are not restricted to software development. All the traditional configuration control practices employed in the development activities(1) should be employed in the maintenance process as well but at a more detailed level. In fact many of the Configuration Management functions do not exist until the maintenance phase has begun. Software Configuration Management is especially important when there are common applications used in many parts of the organization. This is because any uncontrolled minor change in the application to suit an individual requirement might well result in major change requests by others.

In addition, Configuration Management avoids the problem of having users report a software errors or request changes in the absolute version of the software.

Software Configuration Management is a term used to cover a collection of techniques(2) that when applied to software development and maintenance will result in improvement in the quality of the software product, reduces its maintenance costs and improve the management function for the development/maintenance processes(3).

MECHANICS OF SOFTWARE CONFIGURATION MANAGEMENT

Software Configuration is built around four basic steps(4) :

1. configuration identification

2. configuration control

3. configuration status accounting

4. configuration validation

CONFIGURATION IDENTIFICATION

Configuration Identification involves specifying and identifying all components of the application system. There are three dimensions to this identification process(5) :

1. The system must be broken down into a number of known manageable modules.

2. The modules must be uniquely identified.

3. As the modules change with time, the various versions that appear must also be uniquely named.

CONFIGURATION CONTROL

Configuration Control imposes the restriction that no change to the specification or program code is allowed without first obtaining approval from the relevant authority (usually the maintenance manager).

The major concern of Configuration Control is to ensure that for any proposed change to an agreed specification, the same degree of consideration and the same acceptance criteria for the initial specification are applied. This minimizes the possibility of what that a minor change (by local considerations) turns out to have major forseen effects on other applications.

CONFIGURATION STATUS ACCOUNTING

Closely related to Configuration Control is the concept of Configuration Status Accounting. This process involves the recording of all current and historical data concerning the program configuration and report this information to the interested parties. The information to be circulated to all parties concerned should include(6) :

1. States of various specification concerns of the application system.

2. Proposed changes to the specification.

3. The implementation status of approved changes.

CONFIGURATION VALIDATION

Configuration Validation involves a series of reviews and audits to ensure that there is conformity of the application to its specification.

This validation and audit usually involves the use of structured analysis techniques such as inspections and walkthroughs(7).

AUTOMATED CONFIGURATION MANAGEMENT

Feldman(8) suggests the use of an automated approach to configuration management. An example of such automated tools is the "MAKE" facility which runs on the "UNIX" operating system. This configuration management tool assists in the rebuilding of complex software system after parts of the system are modified. The tools achieve this by keeping track of the relationship between the parts of the systems and then issue the commands

needed to make the parts consistent after the changes are made.

REFERENCES

1. Glass R.L. and Noiseux R.A.
   "Software Maintenance Guidebook"
   Prentice-Hall NJ 1981 pg 154 - 155

2. McCarthy R.
   "Applying the Techniques of Configuration Management"
   Defense Management Journal 11, No 4 1975

3. Buckle J.K.
   "Software Configuration Management"
   The MacMillan Press HK. 1982 pg 1 - 8

4. Buckle J.K. - Op Cit pg 9 - 20

5. Buckle J.K. - Op Cit pg 33 - 40

6. Buckle J.K. - Op Cit pg 40 - 48

7. Bersoff E.H. and Others
   "Software Configuration Management : A tutorial"
   Computer Jan 1979

8. Feldman
   "Make - A Program For Maintaining Computer Programs"
   Software Practices and Experience April 1979 pg 225-265

5.5.1.4 CONSULTANT MAINTENANCE PROGRAMMERS

The DP manager faces a dilemma in the maintenance function. On one hand computer personnel are not interested in becoming maintenance programmers and on the other hand, delaying user change requests could damage goodwill between the DP department and the end users.

In addition, as a the result of allotting a fixed number of DP personnel to maintenance activities, the DP manager has difficulty in smoothing the maintenance curve when it is at its peak without adversely affecting response to the user change requests.

To minimize the above problems, Mcgregor(1) suggested the use of consultant maintenance programmers.

TYPES OF CONSULTANCY SERVICE

Basically, the alternatives open to the DP manager in using consultant programmers are(2) :

1. The day to day maintenance requirements are performed by company computer personnel but the services of consultant programmers are engaged to handle major maintenance.

2. All maintenance work is performed by consultant programmer.

3. Part of the normal maintenance routine is performed by the company staff and the consultant programmer service is used to do the balance of the normal routine maintenance and "peak" requirements.

THE ADVANTAGES OF USING CONSULTANT MAINTENANCE PROGRAMMER

There are many advantages in using consultant programmers for the maintenance function. These include :

1. Reduce costs.

In cases where maintenance activities fluctuate widely, these may be reduced costs in using consultant maintenance programmers.

2. Improved productivity.

Unlike typical DP staff, consultant maintenance programmers are trained in their field and require less training to acheive a given level of efficiency.

In addition, the use of an outside consulting service would enable the DP staff to acquire a higher level of skill by learning the

latest maintenance techniques from the consultants.

3. Reduced investment costs.

Another advantage of using a consultant programmer service is that the company can minimize its investment on maintenance facilities such as building, software tools (in some cases) and hardware relating to the maintenance function.

4. Improved staff morale.

Relieving DP staff of "unwanted" maintenance activities can result in improving staff motivation and hence may reduce staff turnover.

PITFALLS

Like other maintenance techniques, the use of consultant programmers has its pitfalls too. These include(3):

1. Consultant programmers may not be able to provide the quality of work that could be expected from in house DP personnel because the latter are familiar with the application systems and the operating environment (eg. hardware, organization procedure and organization objectives). This is particularly so for application system which is tailored to an organization and cannot be easily learnt from the existing documentation.

2. The consultant programmer may not be able to cope with emergency critical maintenance especially in the early stages of engagement where he/she is still not familiar with the existing system.

3. The danger of lack of continuity exists because the company might not always be serviced by the same consultant programmers.

4. The consultant programmers do not have the same sense of loyalty and responsibility that in-house normally have.

5. The use of consultant programmer is expensive.

PRACTICAL CONSIDERATIONS

The steps in using consultant programmers for the maintenance function can be summarized as follow(4):

1. Cost analysis

The DP manager must first make a comparison of the advantages consultant programmers over having permanent maintenance staff. Caution has to be exercised to ensure that both tangible and

intangible considerations are included in the analysis.

2. Consultant programming service

Upon deciding to use a consultant programming service, the DP manager has to appoint a maintenance manager/supervisor whose responsibility involves scheduling and planning consultant programmer involvement with the maintenance process. He/she also acts as the liason officer with the user departments.

3. Development of the maintenance procedure

This should include the scope of commitment, responsibility and authority of the consultant programmer. The key point here is to avoid duplication between the staff support function and the function of the consultant programmer.

4. Training and familiarization

Conducting training and familiarization for the consultant programmers, as necessary.

SELECTION OF CONSULTANT PROGRAMMER SERVICES

The following main points should be considered when choosing between alternative consultant programmer organization(5):

1. The integrity of the consultant organization.

2. The expertise of the consultant organization in their field.

3. What the consultant organization can offer in terms of their staff

    a. adequacy of DP staff

    b. sufficiency of skilled DP staff

4. How completely the contract service meets the maintenance requirements, in terms of necessary software tools and hardware facilities.

As a general rule, a good consultant programming organization is one which has been seen to be able to consistently perform quality maintenance for most of its clients.

The final decision on whether to use consultant programmers for the maintenance function depends on cost savings. For a company which does not have the necessary maintenance expertise or for which it is not economic to set up a permanent maintenance organization, the use of consultant programmers seems to be a promising proposition.

REFERENCES

1. McGregor B.
"Program Maintenance"
Data Processing May - June 1973 pg 173

2. Mann L.
"Maintenance Management"
Lexington Books 1976 pg 249 - 250

3. "Program Maintenance : User´s Views"
Data Proccesing Sept - Oct 1973 pg 314 - 317

4. Jordon J. H.
"Research In Contract Maintenance"
Techniques Of Plant Engineering And Maintenance Vol XIX
pg 253 - 254

5. Mann L. - op cit pg 255 - 256

5.5.1.5 MAINTENANCE EDUCATION PROGRAM

Maintenance education forms an integral part of the maintenance strategy to improve maintenance productivity. Although many DP organizations have some sort of training schedules, they are done on an irregular basis and often without a well planned program. There are two aspects of maintenance education.

1. user education program

2. maintainer education program

USER EDUCATION PROGRAM

User familarity with the existing system plays an important role in the maintenance process. This importance is reflected by research done by Lientz and Swanson(1) which revealed that over 40% of maintenance problems have user ignorance of the current application system as a factor.

The objective of the user development program is to increase user knowledge and sophistication in the use of the appliacation system.

DESIGNING A USER EDUCATION PROGRAM

Traditionally, a user education program is regarded as a task to be undertaken prior to the implementation of the application system. This policy is unsatisfactory as it fails to recognize the effect of user turnover which could undermine previously established familarity and experience(2). A user education program should be developed to encompass and support the full life cycle of the application system.

Designing the user education program involves the following activities(3).

1. The establishment of a standard classroom system familarization package.

2. The design and implementation of a tutorial function within the application itself.

3. Job rotation and enlargement for selected users to provide multiple-role experience with the application system.

MAINTAINER EDUCATION PROGRAM

Management often has a tendency to ignore maintainer education and assumes that new and inexperience programmers will become proficient in their job through some nebulous on-the-job attachment to an experienced programmers(4). This misconception can be costly in terms of maintainer productivity, especially if the supposed-to-be-experienced programmers themselves didn't receive much formal training. Furthermore, there are other training methods which may be more effective in training new and existing maintainer programmers.

STEPS IN A MAINTAINER EDUCATION PROGRAM

The initial step in designing any maintainer education program is the identification of the need of training. This can be achieved in the following ways

1. Use of a Standard

By comparing the maintainer productivity with a standard (either published or an in-house established standard) DP management is able to determine whether the maintainer programmer is below standard and hence whether a training program is necessary.

2. Indication from Project Supervisor

This is an indication from the maintainer project leader as to the need to retrain existing programmers. Retraining is important especially when the DP profession is evolving at such a rapid rate resulting in many maintenance techniques becoming obsolete and at the same time introducing many more effective techniques.

TYPES OF EDUCATION PROGRAM

On the job training seems to be the most popular type of education program. However, care must be exercised in choosing the teacher-programmer. The ideal teacher-programmer should be proficient both in technical and communication skills, and preferably has undergone a formal training course. Local universities and technical colleges are the next source of training for maintainer programmers. However, such training has to be supplemented by on-the-job training as these institutes only provide basic maintenance training, if indeed they provide any.

Another source of education for maintainer programmers is through seminars and short courses. These courses are offered by commercial DP consulting firms, universities and professional societies such as New Zealand Computing Society . The attractive features of this option are that not only does it give maintenance programmers an opportunity to improve their

individual performance but also to ascertain the opportunities that are available to improve the entire maintenance system.

DESIGNING A MAINTAINER EDUCATION PROGRAM

Designing a maintainer education program involved the following steps(5).

1. Define the topics to be covered in the program. These are usually identified in meetings with DP management, project leader and maintenance programmer.

2. Determine who will conduct the training program. The DP management has to decide which type of training program mentioned earlier is suitable for the training course.

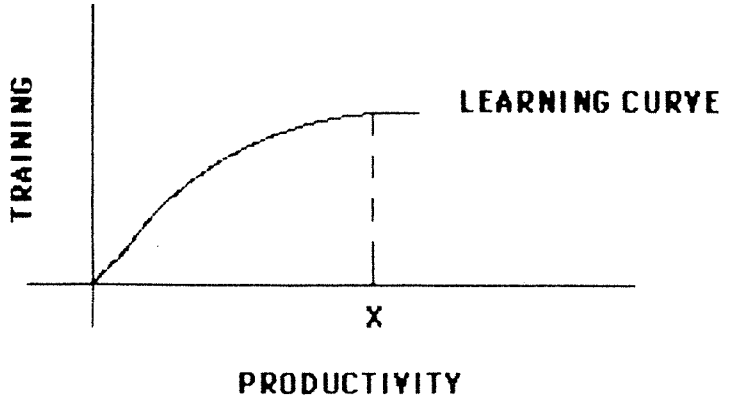
3. Develop the Detailed Course Framework. This includes establishing detailed course outlines and determining who should participate in the program.

4. Establish all educational resource materials required to run the training program (eg computer time).

5. Finally, an assessment of the training program. This can sometimes be done by comparing the before-training and after-training productivity index.

Educating the end-user and the maintenance programmer is part of an effective maintenance management strategy. It is a means to impart to those without experience a form of "second-hand" experience in a shortest possible time. However, care must be taken to ensure that the DP management is not over zealous in the maintenance education program. A learning curve similar to that shown in figure 5.5.1.5 should be plotted. Ideally the education program should cease at point X shown in the graph where maintainer and user productivity index is at its peak.

FIGURE 5.5.1.5



REFERENCES

1. Lientz B.P. and Swanson E.B.
"Software Maintenance Management"
Addison-Wesley Publishing Company 1980 pg 115 - 148

2 Lientz B.P. and Swanson E.B. - Op Cit pg 172

3. Lientz B.P. and Swanson E.B. - Op Cit pg 172

4. Mann L.
"Maintenance Management "
Lexington Books 1976 pg 237

5. Mann L. - Op Cit pg 241 - 243

5.5.1.6 SCHEDULED MAINTENANCE

Scheduled maintenance is a general maintenance management technique. Rather than performing the change requests at the time they are submitted to the DP department, the change requests are implemented according to a predetermined schedule eg all change requests submitted between the 1 Jan'X - 30 Mar'X are delayed until 31 Mar'X for evaluation and implementation.

THE MECHANICS OF SCHEDULED MAINTENANCE

The steps in scheduled maintenance can be summarized as follows(1) :

1. establish a schedule of maintenance dates (eg quarterly)

2. all user requests within the scheduled period are accumulated and consolidated into a list

3. at the predetermined date, a consolidated change request list is circulated to the heads of the relevant user departments for reconsideration

4. the modified change request list is returned to the DP department for feasibility study

5. the outcome of the feasibility study is reported back to the users together with all necessary recommendations to make the change requests more viable

6. finally, with agreement between the DP department and the user, the implementation of the users' requests begins

Note that emergency repairs and changes are exceptions in this arrangement. Emergency repairs and changes are performed as they are needed. However, it can be envisaged that with proper schedule maintenance planning, such emergency requests will only constitute a small percentage of the maintenance tasks.

ADVANTAGES OF SCHEDULED MAINTENACE

There are many advantages of adopting the scheduled maintenance technique. These include(2,3):

1. Consolidation of Change Requests

Under the scheduled maintenance policy, the software maintainer is able to consolidate change requests concerning a particular program module and perform these modifications together. This results in higher maintainer productivity as testing and

documentation updates can also be done together.

2. Cost Savings

Cost savings can be substantial using scheduled maintenance. This is because general fixed overhead costs of maintainer familarization on a particular program module are spread among all change requests on that module. This results in lowering the maintenance cost for each change request.

3. Stable Change Requests

By allowing a time period between the initialization of the change request and its implementation, an opportunity is provided to the users to review their change requests. This should minimize the number of situations where a change request is submitted to correct an earlier unplanned one.

4. Thorough Maintenance

Batching all change requests according to application program gives the software maintainer the opportunity to evaluate the interaction between the change requests.

5. Improved Software Development and Maintenance Planning

With maintenance scheduling DP management is able to plan new software development projects for least impact on maintenance functions.

6. Enhance Programmer Motivation

The programmer's responsibility for maintenance is limited to specific period of the year. In other offpeak periods, they can be rotated to be involved in new software development. This enriches the maintainer's job and hence increases his motivation.

However, care must be exercised when implementing the scheduled maintenance policy. An agreement between the DP department and the end users on the schedule timetable must be established. Failing to do this can result in users accusing the DP department of practising delaying tactics on their change requests.

REFERENCES

1. Lindhorst W.M.
"Scheduled Maintenance of Applications Software"
Datamation May 1973 pg 64 - 67

2. Taute B.J.
"Quality Assurance and Maintenance Application Systems"
AFIPS National Computer Conference, 1983 pg 126

3. Lindhorst W.M. - Op Cit pg 64 - 67

5.5.1.7 MAINTENANCE MONITORING SYSTEM

The operational environment of the maintenance process function is not static. Rather it changes with the current state of the Art of DP.

Effective maintenance management must be responsive to such changes. This can be achieved by constantly monitoring the maintenance process and amending the maintenance strategy/decision according to feedback from the maintenance process.

IMPORTANCE OF MAINTENANCE MONITORING

There are many reasons why DP management should be concerned with maintenance monitoring. These include (1) :

1. Management Responsibility

DP management has a responsibility to ensure that the maintenance process function is carried out effectively and efficiently. Such assurance can only be achieved by continually monitoring the maintenance environment and taking corrective action once any weakness is detected.

2. Enhancement DP-User Relationship

A major beneficial effect of an effective maintenance monitoring system is that it results in fewer program "breakdown". This will enhance user confidence on the DP function and hence improve relationships between the end user and the DP department.

3. Save Costs

Effective maintenance monitoring system not only detects existing problems in the maintenance process but also potential problems. It acts as an early warning system to DP managment before a maintenance problem becomes too severe and costly to correct.

4. Maintaining Standards

By constantly reviewing the maintenance function, and removing any weaknesses detected, DP management can be assured that the maintenance process is always up to a certain standard.

OBJECTIVES OF THE MAINTENANCE MONITORING SYSTEM

The objective of maintenance monitoring can be divided into short and long term objectives(2).

Short term objectives relate to the maintaining of the integrity of the operational environment while long term objectives are concerned with improving the controls within the application as well as the controls governing the maintenance of those applications. Long term objectives are important because they ensure that action is taken to establish an operational environment that not only performs the normal monitoring of defects but also tends to prevent maintenance problems from occurring.

THE MONITORING PROCESS

The maintenance monitoring process can be divided into the following steps(3) :

1. Set Monitoring Objectives

This entails identifying the various aspects of the maintenance process that DP management is interested in. An example of such an objective is a monitoring system documentation process.

2. Established Standards of Performance

Once the target to be monitored is known, the next step is to develop a set of measuring yardsticks (standards) which measure its attainment in the maintenance process. An example of standards of performance is a set rules to be observed in the documentation process.

3. Design Feedback Mechanism

The feedback mechanisms are those in which the data in the monitoring process is captured. These monitoring technique are described later in this chapter.

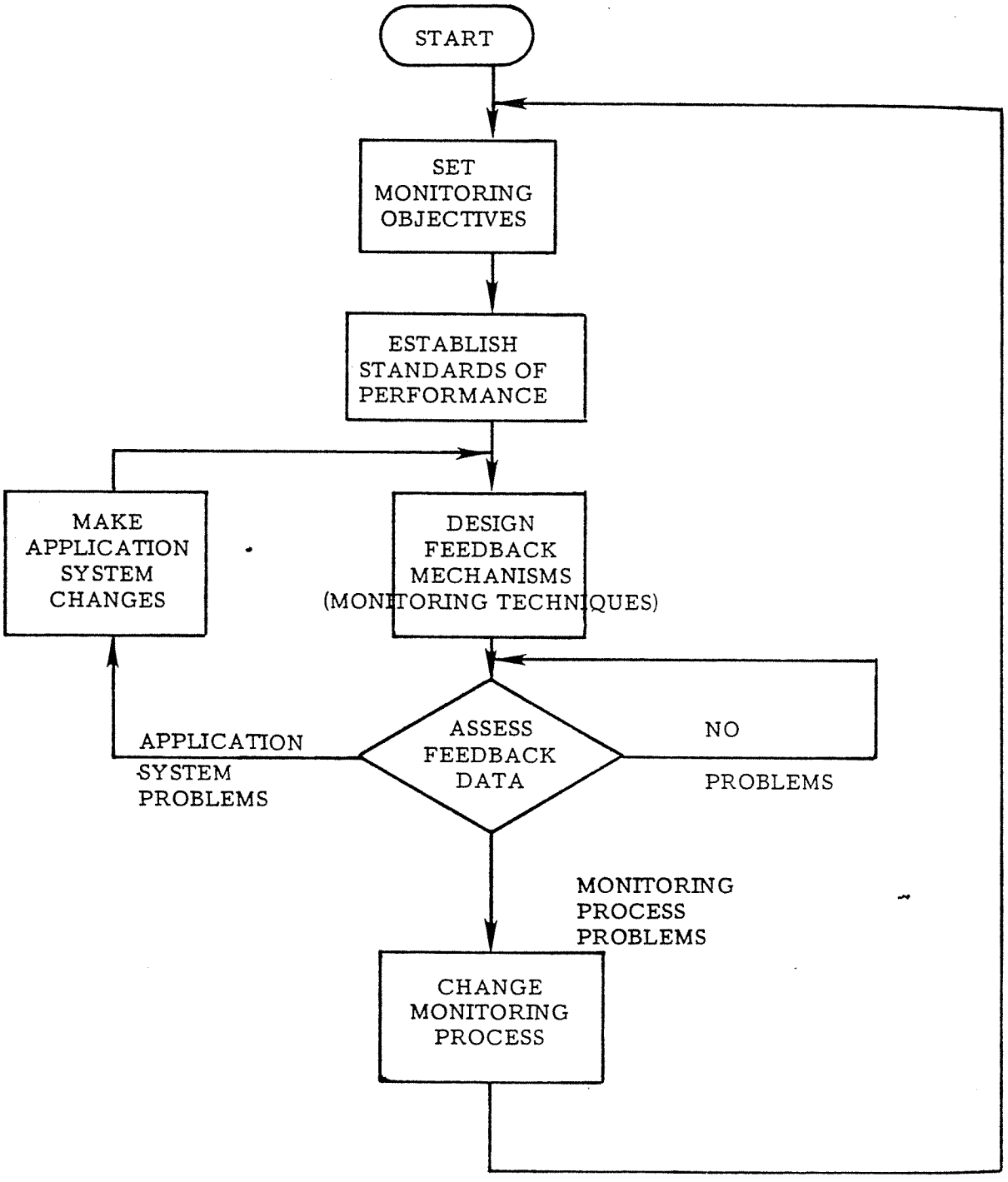
4. Assess Feedback

The information collected in the previous step is compared with the standard of performance established in step 2 above. Based on the outcome of this comparison, DP management can then make decisions as to whether corrective action is necessary.

A diagramatic representation of the monitoring process is shown in figure 5.5.1.7

FIGURE 5.5.1.7

MONITORING PROCESS



Adapted from Perry W. E. - Op Cit pg 322

MONITORING TECHNIQUES

Basically monitoring techniques can be categorized into 3 major groups

1. compliance with standards
2. maintenance management information systems
3. functional structuring

COMPLIANCE WITH STANDARDS

This is an audit-related function of the DP management. Information is gathered mainly through physical observation, examination and testing procedures. This inspection of the maintenance process is usually done periodically (eg monthly)

MAINTENANCE MANAGEMENT INFORMATION SYSTEMS

The maintenance management information systems mentioned in chapter 5.3 are an important means of gathering information for the monitoring process. For instance, program metrics (an output from Maintenance Management Information System) can be used as an indicator of adherence to the standards set in step 2.

FUNCTIONAL STRUCTURING

An effective method for monitoring the maintenance environment is to structure the maintenance team such that the activities of one person acts as a crosscheck for another person. For example, separating the task of coding and testing. This technique is widely adopted in the acccounting environment.

REFERENCES

1. Perry W. E.
"Managing Systems Maintenance"
Q.E.D. Information Science Inc 1981 pg 317-318

2. Perry W. E. - Op Cit pg 320

3. Perry W. E. - Op Cit pg 320-323

5.5.2 PREVENTIVE MAINTENANCE METHODS

5.5.2.1 SOFTWARE MAINTENANCE WITH FOURTH GENERATION LANGUAGES

Software maintenance problems have been a well-known stumbling block for software development because of the amount of resources they consume. Research done by Coble(1), Batt(2) and Stamps(3) showed that the backlog in software maintenance is as high as 2-1/2 years for some organizations. One suggested method of reducing this massive software maintenance is the usage of Fourth Generation language in new software development and also in subsequent maintenance.

Fourth generation languages were created in response to deficienciesy of third generation languages. Third generation languages such as Pascal and Cobol have a major disadvantage in that they require extensive coding for simple commercial applications. Moreover, debugging programs writteh in third generation languages is time consuming and modification is usually difficult.

OBJECTIVES OF FOURTH GENERATION LANGUAGES

The objectives of fourth generation languages from the maintainer´s viewpoint are :

1. to enable a rapid and easy modification of application program thereby reducing maintenance effort

2. to generate bug-free code from a high level expression of requirements

3. to make languages easy to use so that end users can perform their own maintenance on application programs

4. speed up the software development process thereby releasing more computer resources for the maintenance function.

CHARACTERISTICS OF FOURTH GENERATION LANGUAGES

For a programming language to be classified as a fourth generation language, it should satisfy the following criteria(4):

1. user friendly

2. easy for both programmer and end user to use

3. has a short learning curve

4. utilizes non-procedural coding

5. ease of maintenance

6. permits automatic documentation

7. has the ability to make intelligent default assumptions about the user's need.

MAJOR CLASSES OF FOURTH GENERATION LANGUAGES

The 3 major classes of fourth generation languages are :

1. Report Generators.

A report generator extracts data from a database and formats it into a report

2. Application Generators.

An application generator permits an application to be developed with minimum programming effort.

3. High Level Non-programming Languages eg. RAMIS and STAIRCASE.

IMPACTS OF FOURTH GENERATION LANGUAGES ON SOFTWARE MAINTENANCE

The impact of fourth generation languages on software maintenance is twofold :

1. Increase In Productivity.

Research done by IBM(5) concluded that the maintenance productivity gain as a result of using fourth generation languages for some cases is as high as 800%.

The main feature contributing to such an enormous increase in productivity is that the language is non procedural. As a result, the maintainer needs only to specify what is to be accomplished without having to specify in detail how to achieve it.

For example with NORMAD, a program to list customers, sorted by customer number, is reduced to the simple statement.

"LIST BY CUSTOMER"

Furthermore these non-procedural languages are capable of making intelligent assumptions. For example, the command "PLOT" in NORMAD will result in the software automatically deciding the best form of plot (eg scales, heading lettering etc) to suit the nature of the variables.

2. User-driven computing with the language itself designed for

end users, the user is often able to use this tool to modify a program. A detailed discussion of this aspect of user-driver computing is presented in section 5.5.2.2 of this report.

PITFALLS

As a result of the ability to spontanenously create application programs, negative effects on the maintenance process can result if there is no proper control over the use of 4GLs.

These negative effects include :

1. users can be over-zealous about creating their own application and do so without concern as to their future maintainability

2. as a result of ad hoc program modification, the documentation can become quickly out-dated

Tinnierllo(6) recommends that the following actions should be undertaken when using 4th Generation Languages as software development/maintenance tools

1. A written policy on responsibility for maintaining programs should be established. This is required because of the diversity of users (both end users and programmers) using the tools.

2. To eliminate the documentation problems mentioned earlier, several documentation practices can be adopted:

   a. Permit the end user to document an application at business level. A business application description is similar to the traditional documentation except it reflects more business function than the technical details.

   b. Establish a documentation standard to which all users of the tools must adhere.

3. A quality assurance group should be set up to ensure that the final user developed/modified application satisfies a set of programming standards. This group should also review the efficiency of the user developed/modified application program to ensure that the application does not unduly consume vast amounts of computer resources.

REFERENCES

1. Cable D. F.
"Fourth Generation Languagea Will Impact Productivity If ..."
Data Management 20, 1982 pg 29 - 32

2. Batt R.
"Fourth Generation Tools Get Mixed Reviews"
Computerworld 16, 1982 pg 15

3. Stamps D.
"Hail 4th Generation Languages"
MIS Week 4, 1983 pg 18 - 19

4. Martin J. and McClure C.
"Software Maintenance : The Problem And Its Solutions"
Prentice - Hall, Englewood Cliffs, NJ 1983 pg 241

5. IBM Brochure
"IMS/VS User Talk About Productivity And Control"
G520 - 3511 - 0 IBM Corporation White Plains, NY 1098

6. Tinnivello P. C.
"Software Maintenance In Fourth Generation Language Environments"
AFIS Conference Proceedings 1984 pg 251 - 257

5.5.2.2 USER-DRIVEN COMPUTING

One of the major factor(1) that contributes to increasing numbers of change requests is the outcome of using the wrong approach in the maintenance function.

Traditional program development/maintenance is too time consuming and not appropriate for many users maintenance. Program development/maintenance in the traditional approach cannot begin unless the end requirement is specified in detail. It fails to take into account that not all end users know what they actually want till they have experience with the first version of the modified software. Applying this rigid development/maintenance program to a dynamic maintenance process will only result in massive maintenance backlog.

A new approach is needed -- user-driven computing.

USER-DRIVEN COMPUTING

Lientz and Swanson(2) differentiate application development/maintenance into 2 classes; processing specification and reporting specification.

Processing specification is mainly concerned with development/maintenance on the processes relating to data collection, data logging, data storage and organization. While the Reporting specification deals with processes concerning output data and information enquiry.

Martin(3) took a similar approach by differentiating the development/maintenance process into 2 categories; user-driven computing and prespecified computing. The importance of this distinction proposed by Lientz and Swanson and Martin is that the reporting specification/user-driven computing should be performed by the end users themselves while the processing specification/prespecified computing should be the responsibility of the DP department.

ADVANTAGES OF REPORTING SPECIFICATION/USER-DRIVEN COMPUTING

There are many advantages of the above arrangement. These include :

1. Minimize User Frustration.

With user computing, the end user can obtain application or amendment to current applications much earlier thus relieving the extreme frustration of having wait for a long period before change requests are implemented.

2. Relieve The Current Maintenance Backlog.

Where user maintenance is actually performed by the users themselves, it helps to relieve the maintenance backlog which is currently 2-3 years for some organizations. In addition, it also frees valuable computer resources for the development of more complex and sophisticated applications, required by the organization.

3. More Thorough Application System.

It can also be argued that user-driven computing will result in a more thorough application system. This is because the end users are the ones who truely understand the subtleties of their own applications and thus avoid the problem of misspecification between the analyst and the users.
Moreover, with user-driven computing, the users have the opportunity to experiment with the proposed application, thereby increasing their understanding of what they really want from the application.

USER-DRIVEN COMPUTING AND PRESPECIFIACTION COMPUTING

Martin(4) described user-driven computing application as exhibiting the following properties:

1. The user does not know what he wants till he sees what he gets.

2. Application maintenance usually involves changing report formats or creating new reports.

3. The user is capable of creating his part of the application without involving the programmer. This is usually carried out with the aid of productivity tools such as report generators and high level languages.

4. The maintenance of the application is continuous and incremental as the user gets more experience with the application.

Decision support systems are examples of applications that fall within the user-driven computing category.

On the other hand, Martin suggested that prespecified computing applications tends to have following properties :

1. Formal requirements specifications are required before any program development can be carried out.

2. A complete development/maintenance cycle as discussed in Chapter 1 is employed.

3. The development/maintenance time is slow and usually takes

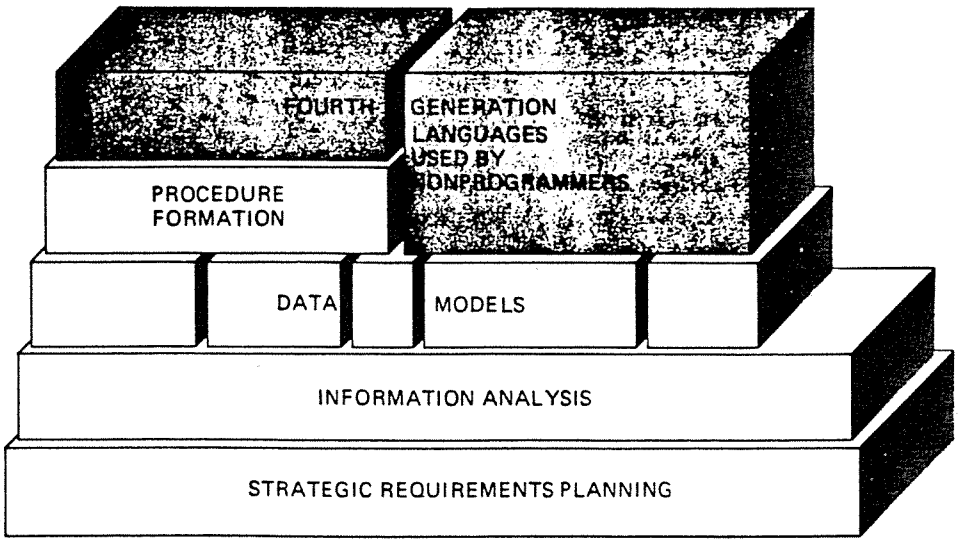
months or years to accomplish.

Examples of applications that can be categorized as prespecified computing applications are compiler construction and airline reservation systems.

NEW APPROACHES

User-driven computing requires new approaches which are different from traditional program development/maintenance. This new approach can be summarized in the diagram below :

FIGURE 5.5.2.2



Information system development in its most powerful form.

Adapted from Martin J. and McClure C. - Op Cit pg 268.

Martin(5) argued that the new approach should support the following facilities :

1. An application generator, a report generator and other 4th generation language components.

2. It should permit software development/maintenance to be carried out quickly.

3. Prototyping will be used to replace the use of a lengthy written requirements document.

4. Application development/maintenance is incremental and iterative as opposed to a single leap forward often associated with the traditional approach.

5. Management control is linked to the data model to prevent spread of incompatible data.

6. The System analyst assumes the role of consultant, helping users create/maintain their own applications.

PITFALLS

The major concerns with the user-driven computing are the problem of programming standards(6). Permitting the end user to create/maintain applications often results in unstructured, complex and inefficient applications.

In addition, there may be compatibility problems with these user-created applications.

Another concern associated with user-driven computing is the long training period before users are proficient in creating their own applications.

IBM(7) suggests the setting up of an Information Center to resolve the above problems.

INFORMATION CENTER

An Information Center is a group within the DP organization whose main purpose is to facilitate, encourage, train and support end users in using the computer for their own applications. It also acts as a watchdog to ensure the application created/maintained satisfy minimum programming standards. The major functions of the Information Center includes :

1. Spread the use of computer facilities to all users.

2. Assist the users in developing/maintaining applications effectively and efficiently.

3. Ensure that data entered or maintained by the users are employed to their fullest potential rather than for personal use only.

4. Ensure adequate accuracy and control on data used by user application.

5. Ensure that the applications developed are auditable and secure where necessary.

6. Link end user activities into the data adminstration process.

REFERENCES

1. Lientz B. P. and Swanson E. B.
"Software Maintenance Management"
Addison Wesley 1980 pg 115 - 149

2. Lientz B. P. and Swanson E. B. - Op Cit pg 171 - 172

3. Martin J.
"Application Development Without Programmers"
Prentice - Hall, Englewood Cliffs 1982

4. Martin J. and McClure C.
"Software Maintenance, The Problem and Its Solutions"
Prentice Hall, Englewood Cliffs NJ 1983 pg 262 - 265

5. Martin J. and McClure C. - Op Cit pg 266 - 268

6. Tate G. and Hayward R. J.
"The Brave New World of nGLs Lessons From DP History"
NZCS papers 1985 pg 325 - 340

7. Martin J. - Op Cit pg 256 - 283

5.5.2.3 RAPID PROTOTYPING

Under traditional software development techniques, program specification must be determined prior to program development. This is considered undesirable where the users themselves are unsure about what they want. Research done by Lientz and Swanson(1) revealed that one of the major causes of maintenance is users' ignorance of their needs. This often results in program mis-specification which is usually corrected at the completion of the development phase (ie. maintenance phase). Correcting specification errors at such a late stage is both time consuming and expensive.

One technique for minimizing this maintenance problem is the use of rapid prototyping.

Rapid prototyping involves building a model of the proposed system to test its specification before committing substantial resources to its final development. The prototype is usually defined to tackle a facet of the application rather than the complete version. This is especially so for large and complex application.

Partial prototyping can exist in many forms and these include (2):

1. Dialogue Prototyping.

Dialogue prototyping is used to simulate the user interface with the proposed system. It allows the users to view what they will be seeing when the final system is implemented. It also permits the user to have first-hand experience of the interface to the proposed application thus giving them the opportunity to contribute ideas for its improvement.

2. Data Entry Prototyping.

Data entry prototyping involves simulating the data entry environment to determine whether it suits the user needs. For instance, in data entry prototyping, the analyst might want to test speed and accuracy of data entry by the user. This yields valuable information on the types of validity checks that should be incorporated to enhance the reliability of the data.

3. Report System Prototyping.

A study by Lientz and Swanson(3) study revealed that enhancement to report formats is the most frequent maintenance within the user maintenance category. Report system prototyping attempts to

minimize this.

Report system prototyping involves generating sample reports to test their suitability in terms of design, content and speed. This is usually acheived using report generators.

4. Conceptual Prototyping.

Conceptual prototyping is used to test the fundamental concepts of the application/change request to determine its necessity and feasibility. It involves the usage of quick-to-implement data management systems, standard data entry screens and standard report formats.

5. Calculation and Logic Prototyping.

This is designed to test the accuracy and consistency of the proposed calculation logic. Calculation and logic prototyping is used most frequently in scientific and financial modelling applications.

6. Database Prototyping.

Database prototyping involves creating a mini version of the final database and subjecting this database to interrogation by the user and analyst for its adequacy. The items to be evaluated include adequacy of the data types, data fields and data organization.

7. Application Package Prototyping.

Application package prototyping is used to test out the functional application for its suitability before linking it to other functional applications. This type of prototyping is commonly used in an application system consisting of many small independent functional programs.

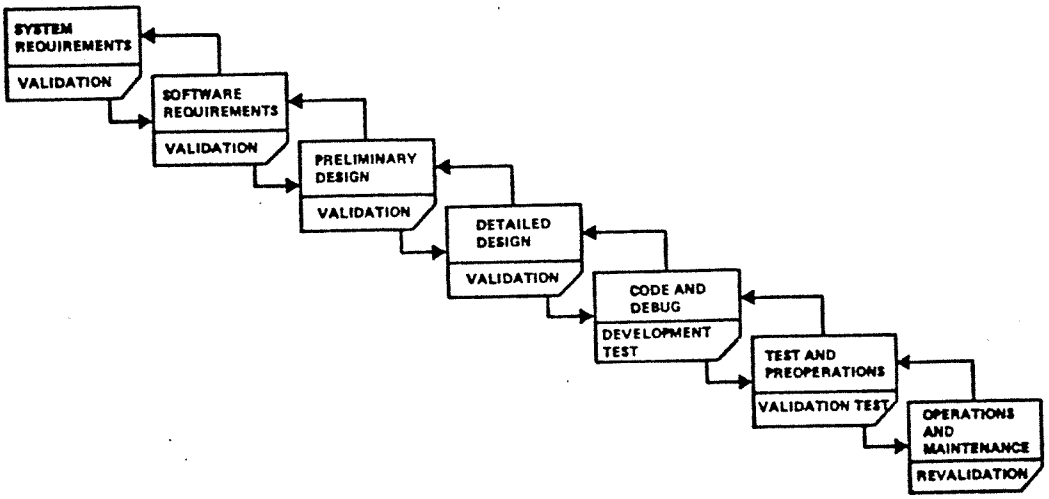
ADVANTAGES OF PROTOTYPING FROM MAINTENANCE VIEWPOINT

One of the major advantages of rapid prototyping to maintenance is that by giving the end user first-hand experience of the sample change request, any misunderstandings between the user and analyst on the change request specifications can be uncovered and resolved. Prototyping permits this to occur at an early phase where the cost involved is minimum. Moreover, prototyping can sometimes be used to demostrate to the user the practicability of the change request and aid the end user to arrive at more feasible change specifications.

STEPS IN PROTOTYPING

Prototyping requires a different software development methodology from that shown in figure 5.5.2.3 A.

FIGURE 5.5.2.3 A



Software life cycle.

Adapted from Boehm W. B.
"Software Engineering"
IEEE Transaction on Computer Vol 25, 12 Dec 1976 pg 1227.

The new approach involves the following steps(4) :

1. Broad Determination Of The User Needs.

For a simple application, this specification can be informal.

2. Creation Of A Prototype.

The emphasis here is to build a model of the proposed system in the shortest possible period. It should only include major functions.

3. Prototype Testing.

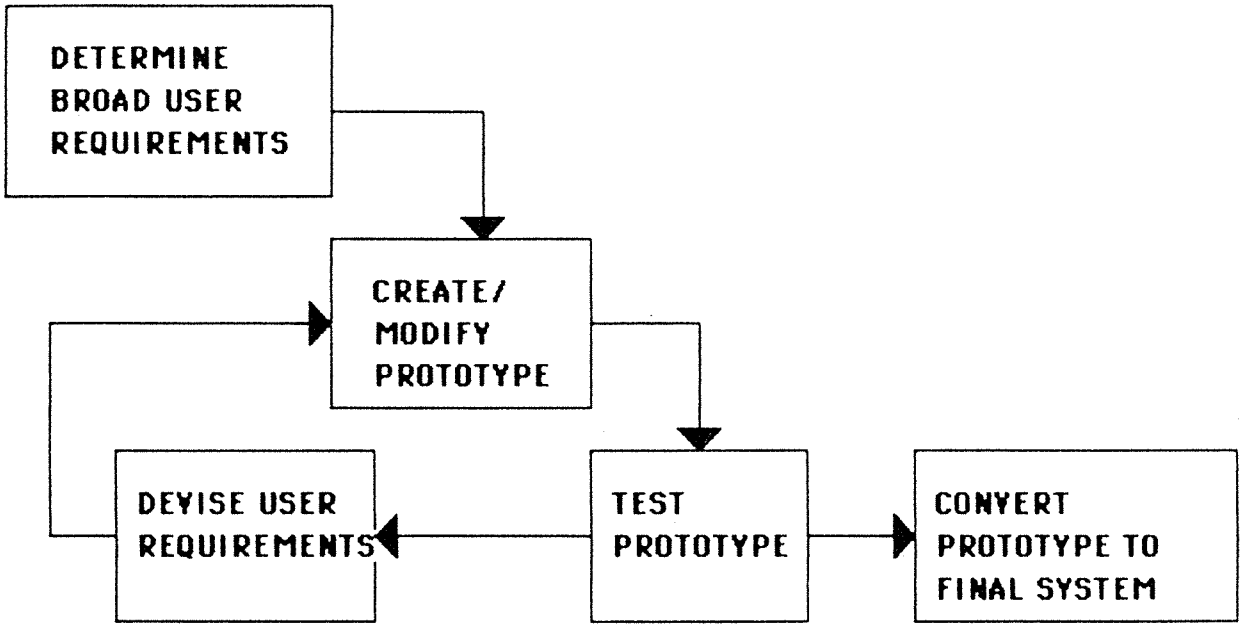
Here the prototype is subjected to testing by analyst and user. Note that step 2 and 3 form an iterative process till a suitable prototype is developed (see diagram).

4. Convert The Prototype To The Final System.

In cases of formal development (usually for large and complex project), the prototype will be used as an application specification for the development process.

FIGURE 5.5.2.3 B

Approach to Rapid Prototyping



PROBLEMS WITH PROTOTYPING

The principal argument against the use of prototyping is the cost of its development. This is especially so for complex and interacting functional modules. Building such a prototype often results in a prototype nearer to the final system rather than a test vehicle. This problem can be partly resolved by ensuring that the prototype developed is used only to test the functional feasibility and by compromising the suitability of the user interaction and reducing reliability and quality standards(5).

In conclusion, rapid prototyping is a valuable technique, not only for new application development but for the maintenance process.

REFERENCES

1. Lieutz B. P.and Swanson E. B.
"Software Maintenance Management"
Addison - Wesley 1980 pg 115 - 149

2. Martin J. and McClure C.
"Software Maintenance : The Problem and Its Solutions"
Prentice - Hall Englewood Cliffs NJ 1983 pg 292 - 293

3. Lientz B. P. and Swanson E. B. - Op Cit pg 67 - 96

4. Martin J. and McClure C. - Op Cit pg 294 - 295

5. Sommerville I.
"Software Engineering"
Addison Wesley 1982 pg 35 - 37

5.5.2.4 SOFTWARE MAINTAINANCE AND THE SOFTWARE DEVELOPMENT PROCESS

In the traditional software development approach, the software maintainer is often not a member of the development team.

This often results in software developed without maintenance in mind. For instance, it is not unusual for the software developer to emphasize the cost-effective aspect of the software and this usually is often at the expense of software maintainability.

There are many advantages associated with the inclusion of a software maintainer as a member of the development team. These include :

1. The maintainer will be equipped to deal with the maintenance of the software. This is especially so for large and complex software.

2. The maintainer can act as a watchdog to ensure that all maintainability objectives set in section 5.1 are adhered.

3. The maintainer can provide valuable information, insights, tools and techniques to the developer. In addition, the software maintainer's previous experience and knowledge can save the developer from reinventing the wheel by using existing components that have been tested and proven reliable.

SOFTWARE MAINTAINER INVOLVEMENT IN THE DEVELOPMENT CYCLE

A typical software development phase consists of 6 stages :

1. requirements analysis

2. specification

3. design

4. coding

5. testing

REQUIREMENTS ANALYSIS

Requirements analysis is a process whereby the requirements for a solution to the program are defined. This stage is crucial to a maintainer because the quality and completeness of the requirements analysis can greatly influence the extent of future corrective and enhancement maintenance needed by the resulting

software(1).

The maintainer at this stage is involved in (2):

1. reviewing the system requirements with the system developers and users

2. considering the impact of new systems on the existing operation environment

3. considering the maintainability when formulating the requirements of the proposed system

4. helping to determine the program priority, enhancement schedules, manpower, and computer resources needed to implement the optional requirements

5. determining what changes are needed to maintain different versions of the system

6. identifing system requirement subsets to allow for system contraction and expansion

SPECIFICATION

The specification phase involves the definition of the functions to be performed by the software.

The specification phase is often neglected by the traditional software developer(3). This is the outcome of programmer anxiety to commence coding and the eagerness of DP management to show tangible results to the end users. Inadequate system specification can have an adverse effect on system reliability and hence increases future maintenance effort.

Software maintener involvement in the specification phase includes assistance, from a future maintenance point of view (4):

1. identifying the subset of functional and data structure components that are required as well as those that are optionally needed in the basic system

2. reviewing specification with users and developers

3. evaluating existing system/modules and selecting appropriate ones to be used in system development

DESIGN PHASE

The design phase is where the algorithms of the software are developed to describe how each specified software system function is to be performed.

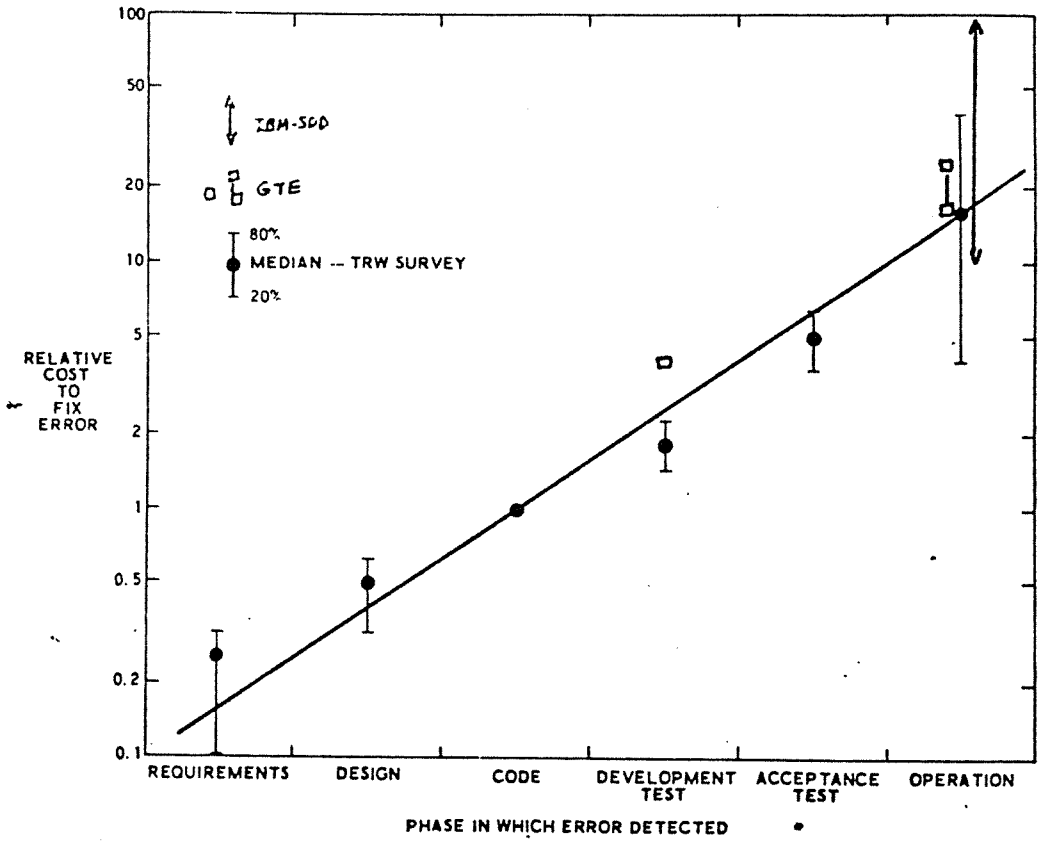
In the traditional software development process, design errors are usually detected and corrected at the testing phase and some at the maintenance phase.

A survey done by Boehm (5) revealed that, of the errors detected during or after acceptance testing, design errors outweighed coding errors by 36 percent.

This is undesirable because the later an error is detected, the more costly it would be to correct.

This is shown in the following graph.

Figure 5.5.2.4



Software validation: the price of procrastination.

Adopted from Boehm B. W. "Software Engineering"
IEEE Transactions on Computers Vol 25, 12 Dec 1976, pg 1228

This problem can be reduced if the software maintainer is involved in the design phase. In addition, the software maintainer, based on his/her previous experience, can suggest ways in which the system can be expected to change. This helps to build flexibility into the software design for future

enhancement.

The software maintainer's role in the design phase include (6), (7):

1. helping to design a flexible software for future enhancement

2. performing change exercises on the design to test its flexibility

3. reviewing the impact of the design on the current systems

4. evaluating the generality of the system design in terms of its ability to operate under different environments (eg. a different hardware configuration)

5. ensuring that the resultant design has the following features:

    a. is capable of isolating volatile functions from other modules

    b. provides module interfaces that are independent to cater for expected changes to the system

CODING

Coding is the process by which the design algorithms are transformed into computer code.

From the maintenance viewpoint, program coding should be readable, understandable and should adhere to coding standards.

The software maintainer involvement in the coding phase includes (8):

1. To review the source code prior to the testing phase to ensure compliance to specification and coding standards.

2. To conduct complexity analysis on the source code to measure understandability.

TESTING

Testing is the process of demonstrating that the software conforms to specification and performs correctly with a wide range of input data.

The software maintainer involvement in the testing phase consists of two parts (9):

1. As a recipient of the information, the software maintainer has access to the test plan and test data. This information can be used for future testing in the maintenance phase. Moreover,

the errors found during the testing phase give the maintainer an indication of which modules are error-prone and hence may require more corrective maintenance in future.

2. As a participant of the testing phase, the maintainer can capitalize on his independent, objective position in evaluating the completeness of the test plan. As pointed out by Myers (10), testing should be done by persons (e.g. software maintenance) other than the system developer. This is to ensure a balance between system reliability and the system efficiency emphasized by the software developer.

REFERENCES

1. Wegner P.
"Research Directions in Software Technology"
3rd International Conference on Software Engineering proc
May 10-12, 1978, pg 243 - 259

2. McClure C. L.
"Managing Software Development and Maintenanace"
Van Nostrand Reinhold Company 1981 pg 42

3. Zelkowitz M.
"Principles of Software Engineering and Design"
Prentice-Hall NJ 1979 pg 2 - 11

4. McClure C. - Op Cit pg 44

5. Boehm B. and Others
"Some Experiences with Automated Aids to the Design of Larger Scale Reliable Software"
IEEE Transactions on Software Engineering SE-1, No1 March 1975
pg 125 - 133

6. McClure C. - Op Cit pg 50 - 53

7. Parnas D.
"Designing for Ease of Extension and Contraction"
IEEE Transactions of Software Engineering
SE5, 2, March 1979 pg 128 - 137

8. McClure C. - Op Cit pg 55 - 56

9. McClure C. - Op Cit pg 62 - 64

10. Myers G.
"A controlled Experiment in Program Testing and Code Walk-throughs/Inspections"
pg 760 - 768

5.5.2.5 STRUCTURED TECHNIQUES

The key point in preventive maintenance methods is to minimize future maintenance through designing more reliable and maintainable software. The preventive method is an up-front approach in which activities begin at the software development stage. An example of such a method is the use of structured techniques.

Structured techniques(1) are a collection of design and programming methodologies aimed at organizing software in a systematic, structured manner. Structured techniques improve software reliability through the use of well defined procedures in the designing and coding phase of the software development. Their use standardizes program documentation to provide order and visiblity. They stress the importance of the analysis and design stages in software development which are vital in influencing future maintenance effort.

MODULARITY

Fundamental to all structured techniques is the concept of modularity. Modularity can be defined as the process of implementing software in small manageable, functionally oriented, independent pieces. These pieces or modules are usually implemented as subroutine or functions or clusters of functions. The modules themselves are the building blocks of the application system. The rules of modularity in programming can be summarized as follows(2,3) :

1. Break down the program into independent modules

2. Organize the program modules to reflect the design process

3. Construct each program module with the following properties :

    a. Each module should have only one entry and only one exit point.

    b. Each module should represent only one logical, self-contained function.

From a maintenance viewpoint, there are many advantages of a good modular design. These include(4,5)

1. If the function performed by a module must change, only that module which requires change needs to be corrected and the rest of the program is unaffected.

2. If a new need for an existing function arises, the current program can invoke it within one module with minimum impact to the rest of the program.

3. If enhancement in program features is necessary, new modules can be created and called upon at the point of need, without affecting the rest of the program.

4. Program testing can be performed on an individual module and once tested, the module can reliably be used elsewhere in the program.

5. Program errors are easier to locate and correct as the correction will often be restricted to single modules.

6. Program efficiency is easier to improve as an inefficient module can be replaced with a more effective one without having the impact ripple through the entire program.

STRUCTURED PROGRAMMING

Program designs can be divided into 2 classes(6) :

1. Monolithic design

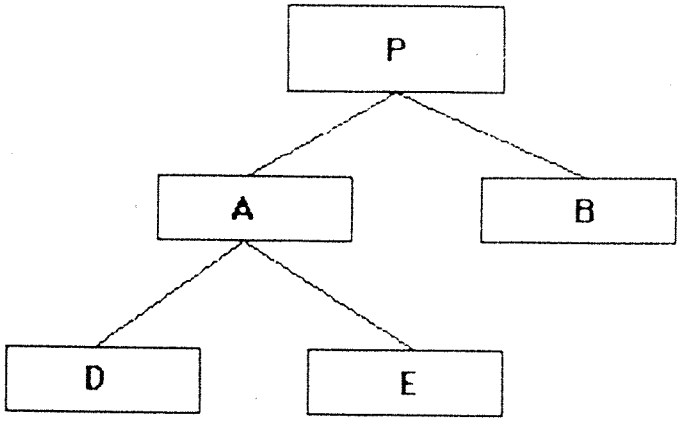
2. Structured design.

Monolithic programming is an undisciplined and non-formalized approach in which the programmer has full freedom on the design of the program. The resulting program structure tends to reflect the programmer's view of the program and is greatly influenced by his experience, knowledge and personal fancy. The final outcome is software that is often unreliable and complicated and difficult to maintain.

Structured programming on the other hand is a formalized approach to program design. Program design follows the principles of hierarchial decomposition and hierarchial structuring.
Basically structured programming can be viewed in two ways; top-down and bottom-up. The top-down design approach starts with a stated specification and is defined into functions. These functions are further redefined into subfunctions and this process of stepwise refinement continues till the process cannot be broken down further - ie at its elementary units.

The top-down design approach can be represented by a typical tree structure shown below.

FIGURE 5.5.2.5



Each box at a lower of the tree represents a refinement of a concept represented by a parent box. For example boxes "D" and "E" together constitute a refinement of the process "A".

Examples of top-down design approaches include :

E. Dijkstra(7)	:	Successive refinement of concepts
N. Wirth(8)	:	Stepwise decomposition
M. Jackson(9)	:	Jackson programming design
IBM(10)	:	Improved programming technologies
ICL(11)	:	Computer aided design system
NCC(12)	:	Program structuring

The bottom-up approach starts from a set of predefined functions and these are combined together to form a program. Returning to the tree example, a bottom up approach would initailly involve identifying the processes "D" and "E" and forming process "A". The process is continued until the top level required is achieved.

ADVANTAGES OF STRUCTURED PROGRAMMING

There are many advantages of structured programming from the maintenance viewpoint. Firstly, structured programming improves program understandability, reliability and modifiability. Secondly, through standardization of programming procedures and style, it improves the ability of programmers to understand programs developed by different individuals.

Finally, program changes will be cheaper as they often only affect a small subset of a programs.

In conclusion, the properties of a well-designed program with maintenance in mind should include the following:

1. consists of modules, arranged in a hierarchical structure (eg. JSP)

2. each module closely adheres to the modularly rules mentioned earlier

3. adequate documentation in each module to describe its function and its interface with other modules.

APPLYING STRUCTURED TECHNIQUES TO EXISTING SYSTEM

Basically there are 4 approaches in introducing structured technique into existing systems. They are :

1. Spare-part approach

2. Structured retofit

3. Code formatting

4. Structured documentation

SPARE-PART APPROACH

The "spart-part" approach was first introduced by Gilb(13,14). Gilb suggests that instead of modifying a program module, we should be replace the whole module with a new one. The advantage of the spart-part approach is that, because the programmer is more familiar with his own work than trying to modify someone else's, less error is likely to result. In addition, this approach may sometimes be used as a gradual transition from the existing monolithic program to a structured one.
Note however, for the spare-part approach to work effectively, there must be adequate documentation on the interface specifications between the modules and the existing program must be modularly designed.

STRUCTURED RETOFIT

Lyons(15) suggests the use of "structured retofit" to convert existing program to a structured program. This approach involves the use of automatic software aids such as code evaluations, reformatters and structure engines to transform the existing monolithic program to a structured one. An example of such tool is Catalyst Corp's COBOL engine.

The key advantage of this approach is that it involves minimum programmer involvement as the bulk of the work is done automatically. However, the success of this approach is also dependent on the quality of the code. It is not much help trying to transform poor unstructured code to a poor structured code. A disadvantage is that automatic restructuring generally increases complexity (e.g. using the McCabe metric).

REFORMATTING

Another approach in introducing structured techniques to existing systems is the use of reformatting tools. Gilb(16) argues that reformatting of the program code is as effective as structured retofit. Moreover, he argues that it is less likely to introduce accidental errors to the system. Basically code reformatting includes the following activities :

1. code indentation

2. standardized labelling conventions

3. limits one instruction to a line

4. standardized use of keywords

STRUCTURED DOCUMENTATION

Even where it is difficult to introduce structured programming into current systems, other structured techniques such as structured documentation(17) be used. Documentation techniques using Hipo diagrams, Warnier diagrams, action diagrams and data flow diagrams are means to this end.

Structured documentation aids in program understanding and hence eases the maintenance effort. A detailed discussion on structured documentation is presented in section 5.5.2.8 of this report.

REFERENCES

1. Yourdon E.
"Managing the Structured Techniques"
Prentice-Hall Englewood Cliffs, 1979

2. Glass R. L. and Noiseux R. A.
"Software Maintenance Guidebook"
Prentice-Hall Englewood Cliffs, NJ 1981 pg 83 - 118

3. Martin J. and McClure C.
"Software Maintenance : The Problem and Its Solutions"
Prentice-Hall Englewood Cliffs, NY 1983 pg 79 - 80

4. Martin J. and McClure C. - Op Cit pg 79 - 83

5. Yourdon E.
"Techniques of Program Structure and Design"
Prentice-Hall Englewood Cliffs, NJ 1975 pg 97 - 99

6. NCC Working Papers
"Program Design Methods"
National Computing Center Limited 1976 pg 25 - 40

7. Dijtestra E. W. and Others
"Structured Programming"
Academic Press London and New York 1972

8. Wirth N.
"Program Development By Stepwise Refinement"
Comminications of the ACM, Vol 14 No 4 April 1971 pg 403 -415

9. Jackson M. A.
"Principles of Program Design"
Academic Press 975

10. IBM
"A Design Aid and Documentation Technique"
GC 20-1851-0

11. ICL
"CADES-Computer Aided Design and Evaluation System"
Computer Weekly July 26, August 2 and August 9, 1973

12. NCC
"Programming Standards Vol 2 Techniques"
National Computing Center Ltd 1972

13. Gilb T.
"Maintainability Is More Than Structured Coding"
in Techniques of Programs and System Maintenance
Lincoln N E 1980 pg 201 - 203

14. Gilb T.
"Spare Parts Maintenance Strategy for Programs"
in Techniques of Program System Maintenance
Winthrop Publishers Inc 1981 pg 213-214

15. Lyons M. J.
"Salvaging Your Software Asset Tools-Based Maintenance"
AFIPS Conference Proceeding Vol 50 May 4/7 1981 pg 337-342

16. Gilb T. - Op Cit (1981) pg 213-214

17. Davis W. S.
"Systems Analysis and Design: Structured Approach"
Addison-Wesley Publishing Company 1983 pg 325-341

5.5.2.6 USE OF DATABASE MANAGEMENT SYSTEM INTERFACE

A major problem facing file maintenance is that a single modification in a file environment can set off a chain of other changes(1). This situation is particularly true for systems using flat file organisation.

PROBLEMS WITH FLAT FILE SYSTEMS

A flat file environment can be illustrated best described with the following diagram

FIGURE 5.5.2.6 A

Program	Data file
Application 1	Data 1 Data 2
Application 2	Data 2
Application 3	Data 3 Data 2

In the above example, application "1", "2", "3" in addition to their own datafile, share a common data file "2".

From a maintenance viewpoint, this type of data environment poses serious problems(2) during file maintenance.

Firstly, any modification in the application program affecting the data structure would mean a new data file has to be created. This is highly undesirable especially with a volatile program environment.

Secondly, because of the interwoven nature of the flat file organization, ie many applications each sharing each other's data files, a trivial change in a datafile could spark off a series of related program changes. For instance, in the above data environment, a change in data file "2" as a result of an amendment application A would also mean that modifications in all 3 programs are necessary in order to be able to access the new version of datafile "2". The situation can be analogous to a bowl of spaghetti. Every time a piece of the spaghetti is pulled, it shakes all the others in the bowl. This situation is aggrevated as the number of application programs increases. Consequently maintenance in such a file environment tends to be difficult and complicated.

USING A DATABASE MANAGEMENT SYSTEM INTERFACE

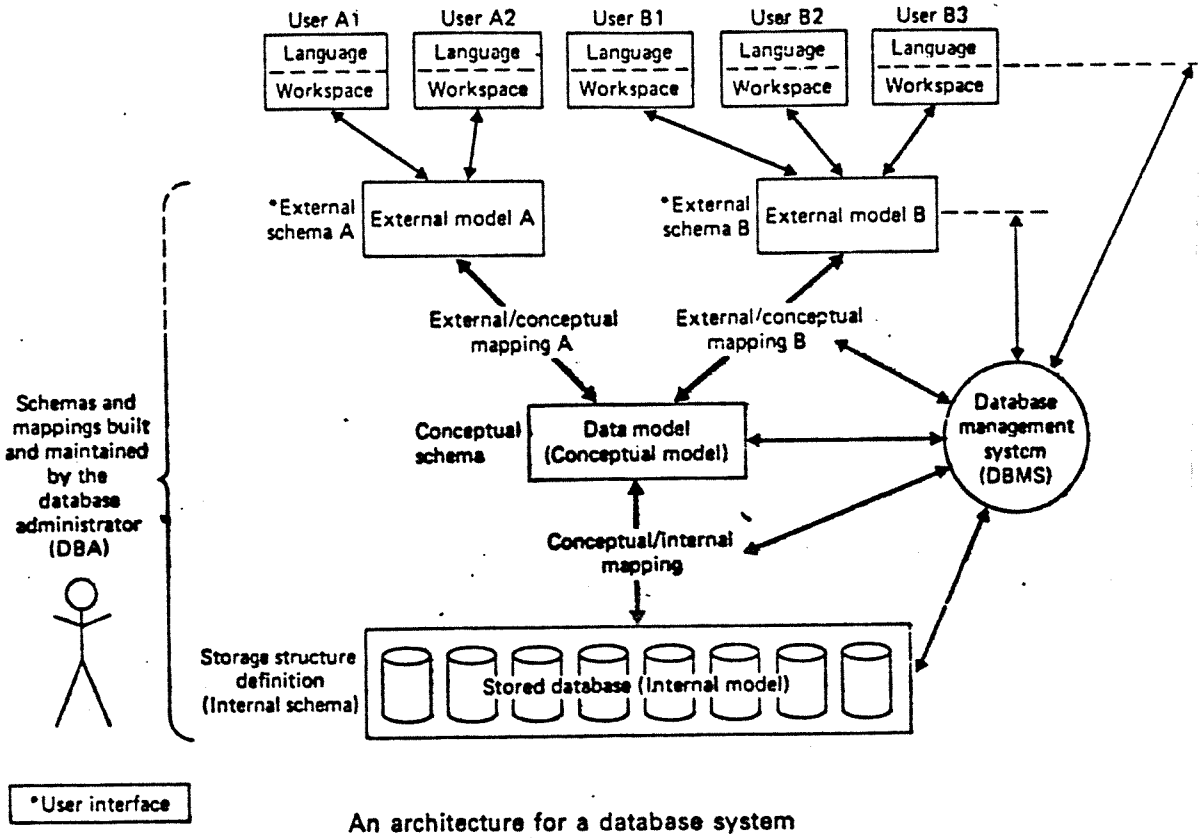
A method of minimizing file maintenance problems is the use of

Database Management System Software. As defined by James Martin, (3) a leading authority : "A data base is a collection of data that is shared and used for multiple purposes." The data is organized in files such that duplication and redundancy are avoided. An important feature of the data base environment is that the data is stored as physical record rather than logical record. (4) Upon invocation, the database management system will search these physical records as requested by the program.

The advantages of this arrangement is that it provids for "data independence " .There exist two aspects of data independence; storage data independence and conceptual data independence.
Storage data independent ensures that user programs can continue to execute correctly whenever changes are made to the organization of the an underlying storage file. Conceptual data independence on the other hand ensures the user program can continue to execute correctly when data fields are removed from or inserted into the conceptual files.

The database management subsystem provides this data independence through a three-level structured database. A three-level database consists of storage file on which conceptual and external files are based(5). A diagramatic representation of the 3-level database is shown in the figure below.

FIGURE 5.5.2.6 B



An architecture for a database system

Adapted from Massey University 58/391/1983 Study Guide pg 52

As such the database environment is especially important for companies where information is a general purpose corporate resource.

TYPE OF DATABASES

Basically databases can be categorized into 3 types(6) :

1. Application databases

2. Subject databases

3. Information databases

The type of database to be adopted in the organization has significant implications for future file maintenance activities.

Under the application database approach, a separate database is created for each new application. This often results in database proliferation, high data redundancy and hence high file maintenance cost.

Unlike the application database, the subject database is created independently of the specific application. For example a product database instead of separate inventory, order entry, or bill of materials databases.

An important advantage of the subject database is that the number of databases in the data environment is much lower compared with that of the application data environment. In addition, the subject database provides a stable data environment on which many applications can be built. An information database has the same advantages of low file maintenence as a subject database but it is organized for searching and fast information retrieval rather than high volume production runs. Such a database is particularly useful for strategic information planning and user-driven computing. An ideal database organization would be a mix of the subject and information databases.

REFERENCES

1. Martin J. and McClure C.
"Software Maintenance : The Problem and Its Solutions"
Prentice-Hall Englewood N J 1983 pg 5-7

2. Martin J. and McClure C. - Op Cit pg 108-113

3. Martin J.
"Computer Data-base Organization"
Prentice-hHall Englewood N J 1977

4. Atre S.
"Data Base : Structured Techniques for Design, Performance and Management"
Wiley-Interscience Publication 1980 pg 18 - 20

5. Bradely J.
"Introduction to Database Management in Business"
Holt-sanders International 1983 pg 125-140

6. Martin J.
"Application without Programmers"
Prentice-Hall Englewood N J 1982

5.5.2.7 STABLE DATA STRUCTURES

The previous section has demonstrated that a system developed in a DBMS environment tends to have lower maintenance cost in the long run.

In this chapter, we turn our attention to an important feature of database systems which minimizes the maintenance effort - data stability.

"Stability" does not mean that the structure of the data never changes but rather that changes can be performed without having to modify other (unchanged) application programs using the changed data set(1).

Databases which are built upon unstable data structures can be compared to building a house without any foundation . Once hit by a change in the file environment , the whole data infrastructure could be damaged resulting in enornmous file maintenance expenditures.

Data stability can be achieved by several means(2). These include :

1. data normalization

2. selection of suitable DBMS software

3. creation of a canonical model

4. end-user participation in the data modelling process

5. stability analysis

DATA NORMALIZATION

Data normalization is a technique which involves examining and grouping of data elements (fields) to form stable structures.

Data is considered normalized if it displays the following properties(3) :

1. it is a logical record

2. the data items in the record contain no repeating groups

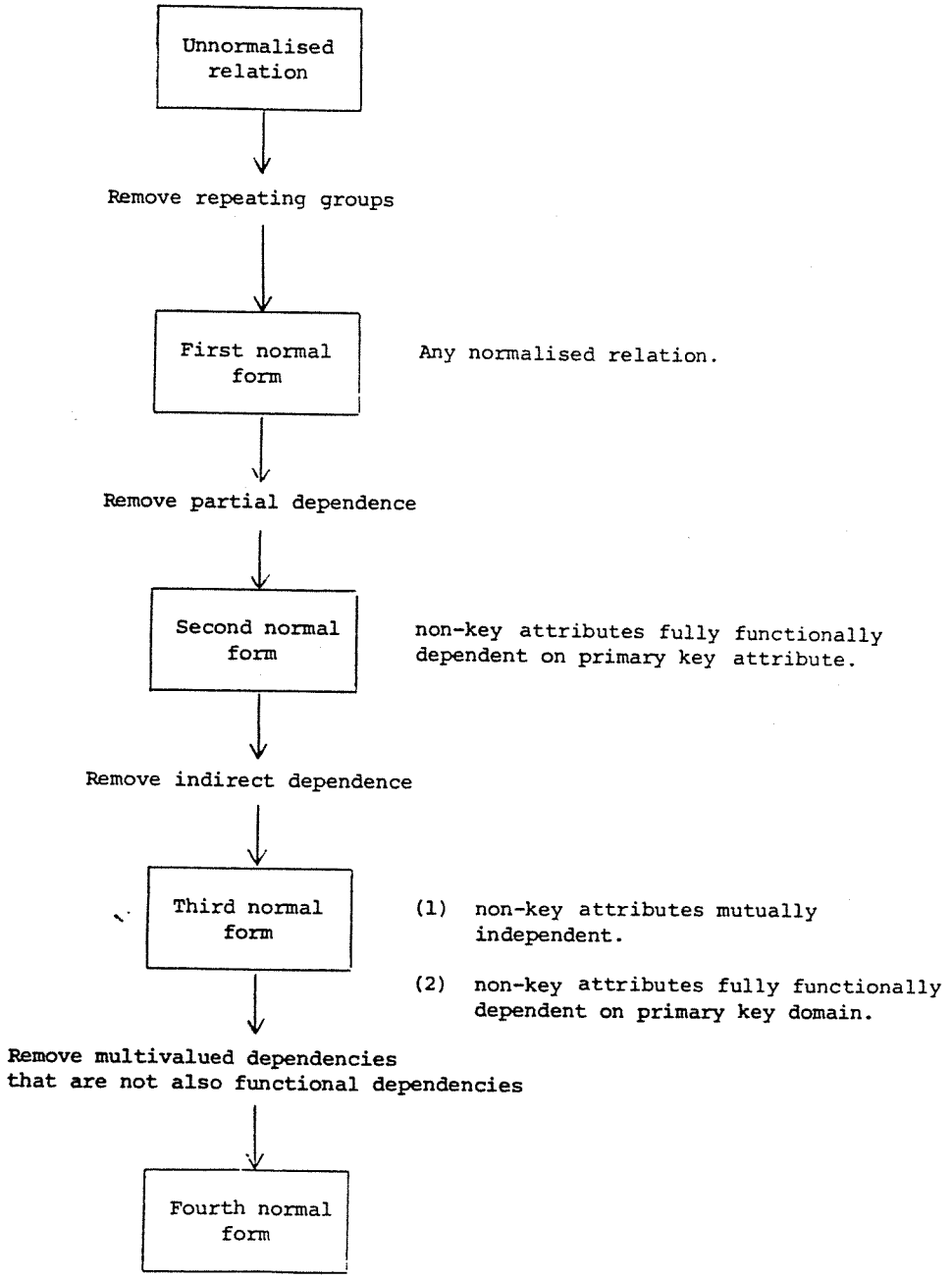
3. it is in third normal form

4. it usually relates to single entity

Normalization Process

The 3-step approach to data normalization is summarized in figure below.

FIGURE 5.5.2.7 A



Adapted from Massey University 58/391/1983 study guide pg 16

The objective of the normalization process is to achieve a data structure that is functionally dependent on its primary key and independent of other fields in the data structure.

Research done by Martin(4) showed that databases with normalized data structures tended to experience lower maintenance expenditure compared to those built upon unnormalized data structures.

SELECTION OF SUITABLE DBMS SOFTWARE

The choice of DBMS software has implications for data stability hence influencing database maintainablity. Martin(5) advocates that to achieved a stable database, the DBMS software must have the following properties :

1. Field sensitivity - the ability to add new data fields to a physical record without the need to alter the logical record used by an application program.

2. Data representation - the ability to represent all required data structures. This includes both network and hierarchical data structures.

3. Secondary keys - the ability to dynamically create secondary key on existing items.

4. Data dictionary - the ability to automatically generate programmer's data from a dictionary. This reduces data incompatibilty.

5. Dynamic relationships - the ability to change associations among records without disrupting existing programs.

6. Distribution independence - the ability to enable the application program to use data even though on a different machine in a network.

7. Support facilities :

    a. flexible query facilities and report generators

    b. application generators

    c. automatic navigation

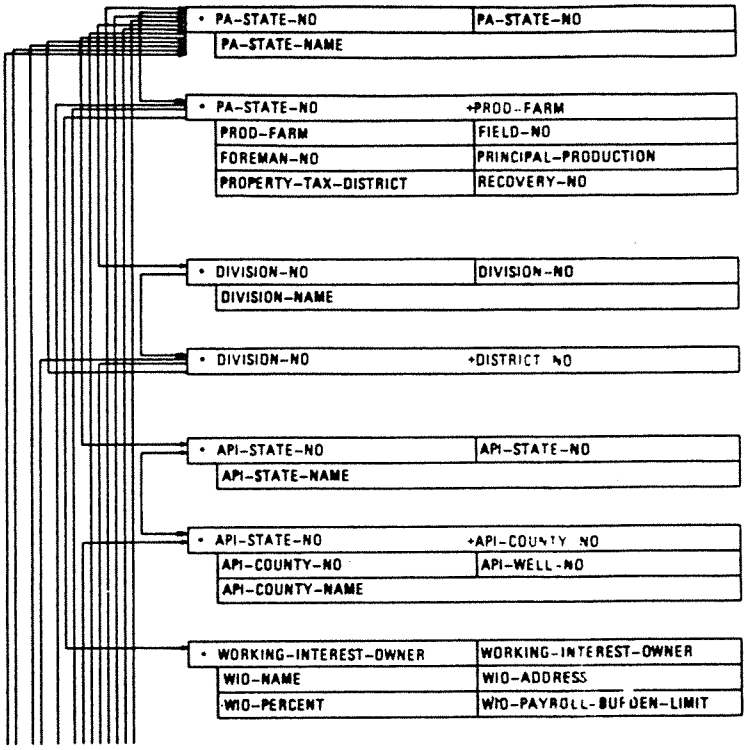
The information database discussed in the previous section has most of the properties mentioned above.

CREATION OF A CANONICAL MODEL

A canonical model is a graph of data which represents the

inherent structure of data and the software or hardware mechanisms which are employed to represent it. It is a graphic representation of the data association in the database in which functional dependencies among the data items are removed.
An example of a canoncial model is shown in figure below.

FIGURE 5.5.2.7 B



Example of the graphic output of a data modeling tool.

Adapted from Martin J. and McClure C. - Op Cit pg 135

A canonical model can be created by using an automatic tool such as "DATA DESIGDNER" developed by Database Design Inc .
A canonical model provides a stable foundation base on which applications can be built.

USER PARTICIPATION IN DATA MODELLING

To be able to create stable data structures, the systems analyst must first understand fully the properties and definition of the data. This can be achieved by involving the end users in the data modelling process.

User participation is important, as often a single item of data may have many interpretations, depending on its functional usage.

For example, in one insurance company(6), the word "claim" actually had six distinct definitions. Unless these data subtleties are fully understood, conflicts in the data structure can cause data instability.

One technique(7) employed to involve users in the data modelling process is the formation of user teams. The members of these teams comprise individuals who are highly knowledgeable in certain areas of the application data. The function of user teams is to check the definition of data in the data dictionary and their functional dependency in the data model.

STABILITY ANALYSIS

Before the database is implemented, certain steps must be taken to ensure that the data are as stable as possible. The steps included in stability analysis are(8):

1. Review the data dictionary and data model to ensure agreement in definition of data elements, and to ensure data requirements can be derived from them.

2. Brainstorm possible future uses of data with the user team.

3. Complete all reverse mappings of any links between keys to identify any possible many to many relationships.

4. Check that any relationship deleted is truly redundant.

5. Evaluate the stability of the candidate key in relation to present and future needs.

6. Convert the data model to a DBMS software schema and validate it by checking that the input view can be devired from it.

Data structures evolve as a company progresses into the future. The objective here is not to create an inflexible database but to create a database that would minimize the maintenance to existing applications as a result of changes.

REFERENCES

1. Martin J. and McClure C.
"Software Maintenance: The Problems and Its Solutions"
Prentice-Hall Englewood Cliffs NJ 1983 pg 135

2. Martin J.
"Managing the Data-Base Environment"
Prentice-Hall Englewood Cliffs NJ 1982

3. Martin J. and McClure C. - Op Cit pg 139

4. Martin J. and McClure C. - Op Cit pg 138

5. Martin J. and McClure C. - Op Cit pg 160

6. Martin J. and McClure C. - Op Cit pg 166

7. Martin J. and McClure C. - Op Cit pg 166

8. Martin J.
"An End-User's Guide to Data Base"
Prentice-hall Englewood Cliffs NJ 1981

5.5.2.8 DOCUMENTATION

Documentation is a critical issue in software maintenance. Inadequate documentation is a major cause of high maintenance costs(1). Incomplete documentation often causes unnecessary debugging problems.

In addition, documentation has a profound impact on software understandability. It is an important communication tool between the software developer and the software maintainer.

PROBLEMS OF DOCUMENTATION

The major problem with documentation is that it becomes out-dated as soon as the software is modified.

This problem is further aggravated by difficulties encountered in maintaining manual documentation. A small change in the software could mean a major revision to the existing documentation.

The other problem associated with the traditional approach to software documentation is the written style. More often than not the documentation is written by the software developer.

Documentation written by programmers is generally sloppy and disorganized(2). The language used often resembles the programming language and there is a lack of coordination within the documentation itself. This causes the documentation to be incomplete, and sometimes conflicting.A host of non-committal statements often results.

Another problem arising from current software documentation practices is that too much emphasis has been placed on the "what" rather than the "why"(3). In addition, the documentation is often written from the programmer's viewpoint rather than that of the user. This is unsatisfactory and can cause technical oversights by the user. The user might submit change requests which in effect have arisen from misunderstanding of the documentation.

The timeliness in the writing of the documentation is also critical. In the traditional approach to system development, the documentation is often done as an after-thought. Consequently, the documentation tends to be simply a process translation and summary. From the maintainer's viewpoint, this is unsatisfactory. Substantial amounts of information generated during the development process is lost. Moreover, the obviousness in the development phase (hence not documented) may be extremely valuable in the maintenance phase.

CRITERIA FOR GOOD DOCUMENTATION

Based on the above problems, good documentation is :

1. easy and inexpensive to produce

2. includes not only "what" aspects but also "why" the system/program/module has been written

3. enhances both the readability abd usability of the software

4. begins as soon as system development commences (ie. at the system specification phase)

SOFTWARE DOCUMENTATION STRATEGY

Software documentation can be divided into 4 categories, user documentation, operational documentation, program documentation and data documentation(4).

User documentation provides the user with instructions on how to use the application system. Operational documentation on the other hand, is directed at the system operators. Program documentation is used by programmer/ maintainer to understand the internal structures of the programs and how the programs interact.

Data documentation is used by the programmer to understand the data and their functional dependencies. This kind of documentation is usually in the form of a data dictionary and a data model.

USER DOCUMENTATION

As mentioned earlier, one of the major problem of software maintenance is the user ignorance of the application system. This ignorance can be reduced by having a quality user documentation.

"Quality" user documentation is one which is complete, current, accessible and is written in a style which the user can associate with(5). To achieve this objective, it is recommended that the user documentation should be written by the user himself or by the technical writer.

There are many advantages associated with "user-written" documentation. Firstly, much of the discrepancies arising from the differences between the system requirement and its function can be avoided. Secondly, the user-manual can be used as an effective familiarization tool for the user.

Alternatively, the user documentation can be prepared by a technical writer. Documentation written by a technical writer who is not computer-oriented is often more readable and

understandable to the user.

PROGRAM DOCUMENTATION

Quality program documentation is an important aid to the understandability of complex software. Glass and Noiseux(6) suggest that quality program documentation should consist of the following elements :

1. Top Level Software Definitions

   a. Overall Structure Summary

      This is a graphic representation of the entire system and its structural components. Examples of such high level documentation are SSA(7) data flow diagrams, ISAC(8) 'A' graphs and JSD(9) system specification diagrams.

   b. Overall Data Base Summary

      This is a pictorial and narrative representation showing the role of data in the total system.

   c. Design Decision Data

      This includes design decisions developed during the development phase.

   d. Underlying Philosophy

      This is a narrative documentation explaining the logic and the purpose of the program.

2. Midlevel Structure

The objective of the midlevel structure documentation is to allow the user to find his way from the top level structure to the detail level structure. Included in such documentation is an index system which ties the overall database, source listing and pointers to the various listing information.

3. Detailed Level Software Definition

Detailed level software definition should be written with maintainability in mind. It should contain comments for the data structures, data base used, functions and implementation anomalies.

The listing should use variable names that are readable and meaningful. In addition, it should be structurally indented. Detailed discussion of these aspects of documentation is presented in Section 5.5.2.9 of this report.

HISTORICAL DOCUMENTATION

In addition, there exists another type of documentation from the maintenance viewpoint, historical documentation.

There are 3 kinds of historical documentation(10) :

1. system development journal

2. error problems history

3. system maintenance journal

SYSTEM DEVELOPMENT JOURNAL

The most important form of the system development journal is the design note, which concentrates on how the software was developed, the design philosophy, the reason for the design selection, the decision-making strategies.

This kind of documentation, usually hand written, is particularly useful to the maintainer when the original software developers are no longer available to explain how and why the software was developed in such a manner. In addition, it may contain a collection of ideas that the maintainer can use for future enhancements.

The system development journal should be filed chronologically and material pertaining to the same subject should be cross-indexed.

ERROR HISTORY

Research(11) showed that parts of the program which were more error prone during the development phase are also likely to be more error prone during the maintenance phase. Therefore, the record of errors and problems faced during the development phase can serve as an indication to the software maintainer as to which program modules are likely to need future modification. In addition, it provides valuable information on the frequency and type of errors that are likely to occur in the program.

SYSTEM MAINTENANCE JOURNAL

The system maintenance journal is different from the system development journal in that the latter records how the system was developed and the former records how the system is maintained.

Like the system development journal, the system maintenance

journal should contain information on the change design, change philosophy, problems encountered during the changes, the implementation strategies and the new version description.

This type of documentation serves as a communication tool for existing program modification and its possible impact on future maintenance.

ON-LINE DOCUMENTATION

So far, our discussion has centred on manual documentation. With the enormous amount of documentation existing today and the number of changes required, especially for large systems, an automatic documentation system would appear to be a more feasible solution.

There exist many advantages of automatic documentation over manual documentation. These include(12) :

1. simultaneous access by several parties

2. changes to the documentation can be facilitated by tools such as formatter and editors

3. control over the documentation can be exercised by using DBMS

4. interactive retrieval by users and maintainers

Examples of such an automated documentation system is "System Information Database" developed by Los Alamos National Laboratory and "MAP" by Amdahl.

REFERENCES

1. Lientz B. P. and Swanson E. B.
"Software Maintenance Management"
Addison-Wesley 1980 pg 115-148

2. Liu C. C.
"A look at Software maintenance"
Datamation November 1976 pg 51-55

3. Liu C. C. - Op Cit pg 52-53

4. Martin J. and McClure
"Software Maintenance : The Problems and Its Solutions"
Prentice-Hall Englewood NJ 1983 pg 177-193

5. Martin J and McClure C. - Op Cit pg 179

6. Glass R. L. and Noiseux R. A.
"Software Maintenance Guidebook"
Prentice -Hall, Englewood NJ 1981 pg 163-167

7. Gane, Chris and Sarson
"Structured Systems Analysis : Tools and Techniques"
Prentice-Hall, Englewood NJ 1979

8. Lundeberg M., Goldkuhl G. and Nilsson A.
"Information System Development - A first Introduction a Systematic Approach"
Department of Information Processing Computer Science, University of Stockholm 1978

9. Jackson M.
"System Development"
Prentice-Hall International 1983

10. Martin J. and McClure C. - Op Cit pg 187-193

11. Martin J. and McClure C. - Op Cit pg 188 - 193

12. Brice L. and Connell J.
"System Information Database : An Automated Maintenance Aid"
AFIPS Conference Proceedings July 9-12 1984 pg 209-216

5.5.2.9 PROGRAMMING STYLE

Readability of a program listing depends on many factors. Programming style is one such factor. There is no one standard programming style but there are a number of guidelines that when adhered to closely will enhance program readability and hence improve program maintainability. Examples of such guidelines are those established by Kerhighan and Plauger(1), Yourdon(2), Myers(3), and Elshoff and Macotty(4). These authors suggested that good programming style should exhibit a common set of properties. They are :

1. adequate program comments

2. meaningful naming of program objects

3. good program layout

4. high-level program control structures

5. consistency in style and program structure

ADEQUATE PROGRAM COMMENTS

Program comments should be regarded as an integral part of the program rather than as secondary statements to the program coding.

Program comments permit the maintainer to acquire a quick understanding of the function of a particular module without an actual walkthrough of the detailed source code. In addition, it provides an explanation of the characteristics of the module, eg. invoking module and parameter passing that may not be obvious in the program code. This is particularly important for large and complex application systems.

Shneiderman(5) suggests using the "chunking" theory in creating meaningful program comments. He argued that by summarizing a group of program statements into a high level semantic language, known as a "chunk", the reader would be in a better position to understand the program listing. This is because the human mind has limited short term memory. By grouping the program codes into a coherent chunk, it will improve the readers' user effectiveness in using their short term memory. In addition, he recommended that the number of chunks in each module should be 7 plus or minus 2.

Elshoff and Marcotty(6) recommend that heading comments should be placed at the beginning of each program. These heading comments should describe the purpose of the program, its external interface and the program mechanics. They further suggest that

the program should be divided into major sections and paragraphs with block comments at the beginning of each paragraph describing its functions.

James Martin(7) recommends that the block comments for each module should include the following information :

1. module purpose

2. effective date (last version)

3. limitations, restrictions and algorithms

4. accuracy requirements

5. input/output

6. assumptions

7. error and recovery procedures

8. information explaining the impact of changes on other modules of the system

Note however, that care should be taken to prevent the over-zealous use of program comments.

Too many comments can result in unnecessary cluttering of a program and hence impair program readability. Sommerville suggests that the number of comments and the information presented in them should be determined by the program size, the complexity of the program and the languages used. He further recommends that these program comments should be closely related to the real-world and should provide an overall functional description of the program units.

MEANINGFUL NAMES

Another important property of good programming style is the use of meaningful names for the program objects (e.g. variables and procedure names).
Meaningful object names enhance program readability and hence ease the maintenance effort.

Sommerville(8) suggests that object names should relate closely to the real-world entities which they are modelled. For example, if a program module is involved in the calculation of a customer statement then the object name should be represented by entities such as invoice-number, customer-number, statement-total etc.

PROGRAM LAYOUT

Good program layout involves the use of paragraphing, spacing and

structural indenting of related statements.

Proper usage of program layout ensures that the program is more elegant and readable.

Blank lines in the program can act as separators which help to distinguish one part of the program from another.

Structural indentation entails a consistent indentation of program statements in the body of a loop and in each arm of a conditional statement.

However, care must be taken to ensure that there is no excessive indentation as it may result in confusion to the reader. Martin (9) recommends a limit of 3 levels to any nesting indentation, but in some circumstances this appears to be unnecessarily restrictive.

PROGRAM CONTROL STRUCTURE

The fourth property of good programming practice is the proper use of high level program control structures.

As a general rule of thumb, the programmer should, whenever possible, use the highest level program control structure available in a language(10). To illustrate this point, assume a situation where the programmer has an option of using a "Do-while" or an "Go-to" structure. Then according to the rule of highest control, the former structure should prevail.

The use of higher-level control structures enhances program understandability.

Sommerville(11) suggests that control constructs should be structured in such a way that the flow of control is strictly hierarchical. In addition there should be a single entry point and a single exit point in each module of a program.

CONSISTENCY AND STRUCTURED PROGRAMMING

Consistency of programming style is another factor that has an impact on program understandability. Consistency in programming style enables the reader to have an idea of what will be coming next thus predisposing the reader's mind to accept the next piece of information.

In addition, programming style consistency allows individuals other than the original writer of the program to maintain the program with less difficulty.

Closely related to program-style consistency is structured programming. Structured programming methodologies eg. JSP(12) ensure a consistent program style and format throughout the

system application. A discussion of these structured programming methodologies is presented in Section 5.5.2.5 of this report.

REFERENCES

1. Kernigan B. W. and Plauger P. J.
   "The Elements of Programming Style"
   McGraw-Hill N. Y. 1984

2. Yourdon E.
   "Techniques of Program Structure and Design"
   Prentice-Hall N. J. 1975

3. Myers G. J.
   "Software Reliability"
   John Wiley N Y 1976

4. Elshoff J. L. and Marcotty M.
   "Improving Computer Program Readibility to Aid Modification"
   Communications of the ACM August 1982, pg 512-521

5. Shniederman B.
   "Software Psychology"
   Cambridge WA, Winthrop Publishers Inc, 1980

6. Elshoff J. L. and Marcotty M. - Op Cit pg 513

7. Martin J. and McClure C.
   "Software Maintenance : The Problem and Its Solution"
   Prentice-Hall NJ 1983 pg 208-209

8. Sommerville I.
   "Software Engineering"
   International Computer Science Series
   Addison Wesley, 1982 pg 118

9. Martin J. and McClure C. - Op Cit pg 205-206

10. Martin J. and McClure C. - Op Cit pg 208

11. Sommerville I. - Op Cit pg 123

12. Jackson M. A.
"Principles of Program Design"
Academic Press, 1975

5.6 MAINTENANCE FUNCTIONAL ORGANIZATION

A major component of effective maintenance management is the establishment of a maintenance function organization. Setting up a maintenace function organization involves 3 important elements(1).

1. the postion of the maintenance function within the organization structure

2. the internal organization(2) of the maintenance function

3. the functions and responsibilities of key personnel

MAINTENANCE FUNCTIONAL ORGANIZATION

Although a high amount of effort is spent on maintenance activities, many organizations fail to recognize the importance of having a defined maintenance structure within the DP organization. This lack of a formal maintenance structure has contributed to the problems of assigning maintenance responsibility, programmer motivation, timely maintenance decision making and communication between maintainer, end user and DP management.

The setting up of separate maintenance function is important not only because it facilitates management control over an area where as high as 75% of the resource is expended but also because it has a direct influence on the resulting maintenance quality. Moreover, such an inclusion in the organization structure would also make the maintenance function more visible to top management who usually have little knowledge of what makes up the bulk of the DP department budget.

MAINTENANCE ORGANIZATIONAL REQUIREMENTS

There are certain organizational requirements a company should adhere to when designing a maintenance functional organization(3).

1. Formalize the maintenance organization so that individual responsibilities and assignments are clearly defined.

2. Establish communication channels and promote interaction between maintenance staff and end users, DP management and development staff.

3. Minimize the dichotomy between the maintainer´s preference to work alone and the need to work as part of a team to produce a coherent and integrated functional unit.

4. Enhance software maintenance visibility so that quality control, standards, software engineering procedures and management monitoring are possible.

5. Promote a maintenance organization structure where state of the art technologies can be mastered and applied.

TYPES OF MAINTENANCE ORGANIZATION

Basically there are 3 ways in which the maintenance function can be represented in the organizational structure.

1. As Part of the Development Function

Here, the maintenance function is incorporated into the development organization. This arrangement is a sort of "time-sharing" basis in which maintenance is provided by the development programmers, who simultaneously are involved in other development tasks.

2. As a Separate Organization

In this arrangement, the maintenance function is a separate entity from the software development organization. It exists purely for the purpose of maintaining software.

3. Matrix Structure

In between the above extremes lies a range of possible matrix structures. An example of such a structure is one where each programmer is rotated around the development and maintenance function periodically. Another variation of the matrix structure is that, when the new system is released to operation, some of the developers "escort" it into the maintenance organization.

ADVANTAGES AND DISADVANTAGES OF A SEPARATE MAINTENANCE ORGANIZATION

There are both pros and cons in regard to the concept of a separate maintenance organization(4).

The arguments advocated by the proponents of the separate maintenance organization include the following advantages:

1. Improves Maintenance Productivity

Studies done by Martin(5) revealed that a substantial number of those organizations that have a separate maintenance organization claim to have increased programmer productivity.

2. Better Management Control

A separate maintenance organization allows management to institute more formal controls on user change requests, maintenance quality and maintenance costs.

3. Enhanced Planning

Separating the maintenance organization enhances manpower planning within the DP department. This is because the DP department is now in the position to develop and adhere to a planning schedule without the conflict of having to split programmer time between new development assignments and maintenance requests.

On the other hand, opponents of the separate maintenance organization argue that such an arrangement can result in serious motivation problems. Their arguments include:

1. Establishment Of A Class System

Establishing separate maintenance staff can result in creating a class system between the programmers. This morale problem is contributed to by the fact that there is a stigma attached to maintenance activities. Details of this stigma are discussed in chapter 2 of this report. Some organizations attempt to solve this problem by offering special incentives to maintenance staff such as monetary incentives, job enrichment through training programs, and opportunities to specialize in a particular maintenance function or application area.

2. Inadequate Resources

For many organizations, it may not be practicable to divide the DP staff into 2 separate organizations due to lack of experienced personnel. Lientz and Swanson in their surveys found that 20% of the DP organizations studied had created a separate maintenance organization and that those that did were usually the larger DP departments.

KEY CONSIDERATIONS IN SELECTING TYPE OF MAINTENANCE ORGANIZATION

The key considerations as to whether a separate maintenance organization is more appropriate depends on the flexibility of the organization's programming staff, and the complexity and importance of the software to be maintained(6).

1. Staff Flexibility

If all the DP staff feel strongly against assignment to maintenance activities, then it is advisable that the maintenance tasks be spread evenly between the staff. If however, some of

the staff enjoy maintenance and are flexible and capable of performing it well, then a separate maintenance organization may be preferable.

2. Software Complexity

If particular software is complex, eg an operating system, then it may be necessary that some or all of the original development team be retained for the maintenance function. In this situation, a "time-sharing" basis or martix structure mentioned earlier would be more appropriate than separate maintenance organization.

3. Software Importance

If the software to be maintained is very important, eg a payroll system, then it is organizationally desirable to separate the maintenance into a separate organization in order to permit rapid and accurate maintenance activities.

INTERNAL MAINTENANCE FUNCTIONAL ORGANIZATION

The proposition of one-man maintenance is no longer considered a feasible structure for most maintenance projects(7). The rapid profusion of software in terms of size, complexity and impact of operation has not only increased technical complexity but also causes difficulty in organizing and managing the maintenance function.

There are many advantages to the concept of a maintenance team. Firstly, researches done by Yourdon(8) and Shneiderman(9) have indicated that maintainers tend to be more effective when they work as a team. For instance, Shneiderman's research revealed that programmers working alone to debug their own program code are the least effective. This is because maintenance programmers have difficulty in detecting their own mistakes. Within a team environment, this problem can be minimized.

Secondly, the team environment often results in an increase in maintenance quality. For example, inexperienced members of the group tend to be motivated to try do their best in order to gain peer approval.

Thirdly, the team concept permits task specialization. This is particularly important for complex software where the maintenance involves a variety of tasks.

Lastly, the team concept can enhance programmer motivation by offering a promotion path within the team.

TYPES OF MAINTENANCE TEAM

There are 2 possible maintenance team structures a company can adopt(10).

1. short-term (temporary) teams

2. long-term (permanent) teams

The temporary maintenance team structure revolves around the concept of egoless programming. It is an informal team which emphasizes the solving of a problem rather than criticizing individual programmers. The short term team is task-orientated, for example debugging a program and conducting a quality control audit.

On the other hand, the permanent team structure is patterned after the chief programmer team concept. Here the main emphasis is on strict organizational structure, discipline, clear leadership and functional separation. The permanent team will continue to exist till the end of an application's useful life.

There are many advantages to the permanent team structure. These include:

1. it permits specialization among program maintainers

2. it minimizes the vulnerability of having to depend on an individual to maintain an application program

3. it improves maintenance quality by providing cross-checks between members of the teams

FUNCTION AND RESPONSIBILITY OF KEY PERSONNEL

The major participants in the team(11) include :

1. Maintenance Administrator

The maintenance administrator is in charge of the maintenance team. His/her responsilities including staffing and promotion of team members and he/she acts as a communication line between the maintenance team and management.

2. Maintenance Leader

The maintenance leader is the most important man in the team structure. He/she is responsible for planning the maintenance activities of the team. He/she must have sound knowledge of project management as well as the application system. An ideal person for the job is someone belonging to the original software development team. Lientz and Swanson refer to such a person as a "maintenance escort".

An suitable candidate is the co-leader of the development team. The benefits of the "maintenance escort" concept are:

1. By working with a member of the original development team, the rest of the team can have first hand knowledge of the application development philosophy and approaches.

2. The effort required to maintain a program varies over time. Therefore by rotating software personnel between the development and maintenance functions, it helps to create flexibility in staffing and eases temporary staffing shortages.

3. It helps to promote communication between the maintenance and development function.

4. Finally, it promotes a sense of mutual respect between the maintenance and development function on technical expertise required in each function. This hopefully would lessen the stigma attached to the maintenance task.

THE CO-LEADER

The co-leader responsibility involves assuming the team leadership when the team leader is not available eg annual leave. The co-leader also acts as the liaison officer between the maintenance team and the development team. His involvement with the development team includes conducting checkpoint quality reviews, defining programming maintainability standards and ensuring that they are adhered to, and gathening all necessary information necessary for future software maintenance support.

USER LIAISON

As mentioned earlier, a substantial amount of software maintenance arises from changes of end user needs.

One method of controlling these ever-increasing change requests is to channel all change requests through user liaison. User liaison acts as a buffer protecting the maintenance team from direct user interruptions. It also helps DP management in identifying long term user needs.

MAINTENANCE PROGRAMMERS

The maintenance programmers are responsible for diagnosing program problems and perform the maintenance task under the direction of the maintenance leader. A maintenance programmer preferably should be an experienced programmer with knowledge of performance tuning and error analysis.

The effectiveness of the maintenance function is greatly affected by its organization - both internally and externally. In this chapter, we have attempted to give an ideal organization structure for the maintenance function. The ideal situation is directed at the larger DP organization, nevertheless the concept is also applicable for smaller DP organizations.

REFERENCES

1. Priel Y.Z.
   "Systematic Maintenance Organization"
   Macdonald and Evans Ltd., 1974 pg 21

2. McClure C. L.
   "Managing Software Development and Maintenance"
   Van Nostrand Reinhold Company pg 165-166

3. McClure C. L. - Op Cit pg 164-165

4. "Easing the Maintenance Burden"
   EDP aNalyser Vol 19, No.8 Aug 1981 pg 1-6

5. Martin J. and McClure C.
   "Software Maintenance : The Problem and Its Solution
   Prentice Hall Inc, N J 1983 pg 422

6. Glass R. L. and Noiseux
   "Software Maintenance Guidebook"
   Prentice-Hall N J 1981 pg 141-143

7. McClure C. L. - Op Cit pg 70

8. Yourdon E.
   "Managing Structured Techniques"
   in Techniques of Program and System Maintenance

9. Shneiderman B.
   "Software Psychology"
   Cambridge, Ma, Winthrop Publisher Inc 1980, pg 86-90

10. Martin J. and McClure C. - Op Cit pg 430

11. McClure C.L. - Op Cit pg 165-169

CHAPTER SIX

THE HUMAN ELEMENT IN SOFTWARE MAINTENANCE

6.0 INTRODUCTION

The human element in software maintenance has been sadly neglected. The recognition of its value and importance to the maintenance task is relatively low.

Both DP management and DP staff hold maintenance work in low esteem(1).

In addition, management rarely recognises the good work done by the maintenance staff.

All these result in low maintenance morale and hence cause sagging and stagnant maintenance productivity(2). In this chapter, the main influences affecting the human element in the maintenance process are analysed.

6.1 MAINTAINER EFFECTIVENESS

The effectiveness of a maintainer can be summarised in the following formula(3);

Maintainer effectiveness = Human Potential x Application factors

The main aspect of the "human potential" element is the suitability of the DP staff for the maintenance task.

The application factors can be further broken down into 2 elements; the morale of the maintainer and the quality of the DP management.

6.2 THE HUMAN POTENTIAL

Apart from the normal tasks performed by the programmer, the maintainer has to deal with a host of problems that are not found in the software development process. To be able to perform the maintenance task effectively, the maintainer must have the following attributes(4):

1. Flexibility

The software maintainer must be able to adapt to a program done by someone else. He must be able to cope with someone else's

programming style and be flexible enough to be able to maintain the application. This is rather difficult because programmers tends to be individualistic as far as programming and documentation is concerned.

2. Broad Background

In addition to being adaptable to a variety of programming styles, the software maintainer should also be well versed in several application developments. This is because a maintainer will rarely be required to only maintain one type of application but also a set of different applications.

Moreover, the maintainer must be able to deal with more than one set of people on the same application. For example, the accountant's emphasis on the inventory program is different from that of the production manager. The former is concerned with the inventory carrying costs while the latter is concerned with material availability.

3. Patience

It is not uncommon for a user to submit a change requests and immediately demanding such changes to be implemented. The maintainer must have patience to deal with such aggresive users rather than provoking an unseemly confrontation with such a user.

Patience is also required for the maintenance activity itself. This is especially true for poorly designed programs or those with inadequate documentation.

4. Self-motivation

An good maintainer is one who is a good self-starter and can function with minimum management supervision.

5. Innovation

An effective maintainer must also be innovative. He must be able to adapt to the current program in its new form and yet minimise maintenance impact on the existing structure. The constraint of working within such a limiting framework is probably the hardest for the software maintainer.

MAINTAINER'S MORALE

There are many ways to improve the maintainer's morale. These can be divided into the following categories(5) :

1. financial incentives

2. non-financial incentives

3. negative incentives

6.3 FINANCIAL INCENTIVES

Financial incentives can be defined as "extra monetary payment" earned for above normal performance(6). This type of incentive is favoured by the companies supporting the scientific management principle. Primarily, scientific management is based on the concept that additional money by itself will improve maintainer motivation and hence better performance.

While financial incentives can never be overlooked as motivators, they must be used with care. Koontz and O´Donnell(7) recommend that if money is to be a kind of motivator, DP management must keep the following points in mind :

1. Financial incentives are more important to a maintainer with urgent money needs. This is particularly so for an employee with a family.

2. Financial incentives should be used as a means of keeping an organization adequately staffed and not primarily as a motivator. This means that the incentive should at no time be used as to correct an inadequate salary structure nor to overcome wages inequities.

3. The financial incentives given must reflect the individual´s performance. This is to ensure that the maintainer is rewarded according to his achievements.

6.4 NON-FINANCIAL INCENTIVES

DP management today must recognise that most maintainers are motivated by the work itself and the way management attempts to make the maintenance task more rewarding and meaningful: that is job enrichment.

A study done by Bergtraun(8) showed that the maintenance staff are more productive and more satisfied with their work when the job is enriched. Herzberg(9), an authority in the area of human motivation, argued that financial incentives are only "hygiene" factors. That is, they do not really motivate, but if inadequate, will result in dissatisfaction. Hertzberg argued that the company should concentrate on the "motivator" factors. Job enrichment is one such factor.

6.4.1 Maintainer Job Enrichment Program

An effective maintainer job enrichment program should address the

following areas(10):

1. The Maintenance Task

The maintenance activities must be planned carefully. There must be a clear job description and the maintenance staff should only report to the DP management and no one else. Maintainer morale often suffers when personnel other than the DP manager give orders.

2. Achievement

The sense of achievement itself is a good motivator. Maintainers are often motivated when given a complex, challenging task and made responsible for their own particular segment of the system. In addition, they like to be consulted on the program they are responsible for the maintenance of.

3. Responsibility

The DP manager should specify clearly what he expects from the individual maintainer's effort in order to achieve overall corporate information objectives. In addition, assignment of responsibility must be proceeded by delegation of authority. The DP manager must ensure that the maintainer has adequate authority (power) to perform the maintenance task effectively.

4. Advancement

Maintainer advancement can be achieved through a meaningful training program. This is discussed in chapter 5.9 of this report.

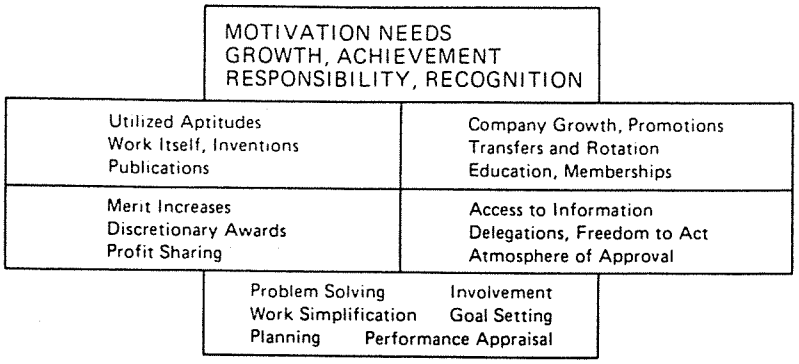
5. Recognition

Recognition must be given to the maintainer for their contribution to the achievement of corporate information objectives. Management tools in these areas include:

1. company notice and newsletters
2. quarterly luncheon for the department
3. promotion to middle management
4. periodic visit by top management eg managing director

A diagramatic representation of the maintainer job enrichment program is shown in the diagram below:

FIGURE 6A



Adapted from Bergtraun E. M. - Op Cit pg 77

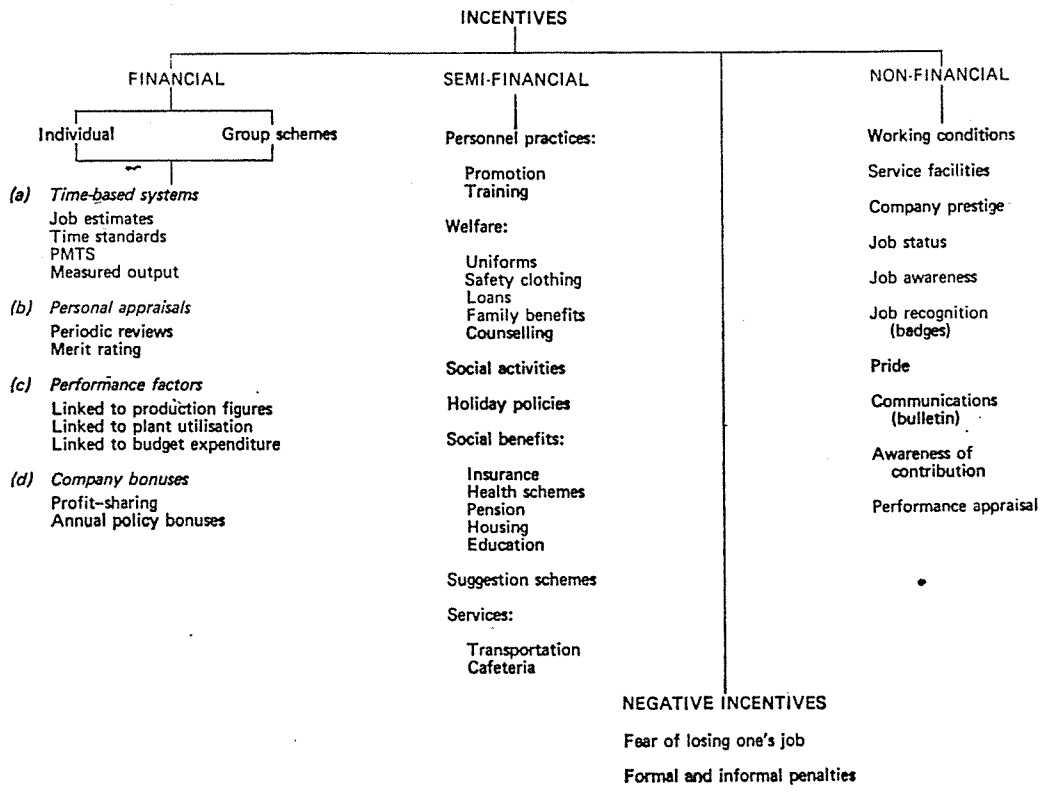
6.5 DISINCENTIVES

Disincentives are used to deter the maintainer from doing certain things. These include the threat of dismissal and salary reduction.

It must be emphasized that disincentives should be the last resort to improve maintenance productivity.

A summary of the major types of incentives mentioned above is presented in the following diagram.

FIGURE 6B



Classification of incentives—financial, non-financial and negative.

Adapted from Priel V. Z. - Op Cit pg 191

6.6 DP MANAGEMENT

DP management efficiency is the next important factor that can influence maintenance productivity.

It is not uncommon for top management to assume that a good programmer is automatically a good team leader and that managerial skill abilities can be developed by just practice and experience(11).

Recent studies in the field of behavioral science prove this assumption wrong. DP management is as much an art as well as a science. There are many elusive and undefinable elements in it, for instance; personal quality, knowledge, skill and experience. It does not only involve the technical-professional aspects but also encompasses the human implications of management.

The table below attempts to list out the main criteria for a good DP manager.

FIGURE 6C

Leadership and managerial abilities required in a maintenance manager.

THE MAINTENANCE MANAGER	
Characteristics of a good leader	+ Traits and skills of a good manager
Personal qualities: Honesty and courage Concentration and persistence Confidence and reliability Initiative and resourcefulness **Human relations:** Courtesy and firmness Helpful and instructive Ability to listen **Leadership 'style':** Team leadership Motivating people Judgment and fairness Power of expression Sense of responsibility	**Organising ability:** Ability to delegate Methodical approach Effective use of time Planning ahead Effective implementation **Analytical ability:** Discerning priorities Problem-solving Decision-making **Management know-how:** Up-to-date techniques in technology and management Communications Utilisation of resources
These characteristics determine how a manager deals with people	These traits and skills reflect his ability to solve problems

Adapted from Priel V. Z. - Op Cit pg 198

In conclusion, maintainer productivity is not only dependent on the technicalities of the maintenance task for example, better maintenance tools, but also entails the human aspects, for example, job enrichment.

REFERENCES

1. Lientz B. P. and Others
   "Characteristics of Application Software Maintenance"
   Communication of ACM Vol 21, No. 6, 1978 pg 466 - 471

2. Pariskh G.
   "Techniques of Program and System Maintenance"
   Lincoln, NB : Ethnotech Inc. 1980 pg 289

3. Priel V. Z.
   "Systematic Maintenance Organization"
   MacDonald and Evans Ltd, 1974 pg 176 - 201

4. Glass R. L. and Noiseux R. A.
   "Software Maintenance Guidebook"
   Prentice-Hall Inc. 1981 pg 21 - 26

5. Priel V. Z. - Op Cit pg 190 - 196

6. Priel V. Z. - Op Cit pg 191 - 192

7. Koontz H. and O'Donnell C.
   "Essentials of Management"
   McGraw-Hill, 1978 pg 427 - 428

8. Bergtraun E. M.
   "Enriching Maintenance Jobs"
   Techniques of Plant Engineering, July 11, 1974 pg 75 - 77

9. Herzberg F.
   "The Motivation to Work"
   John Wiley N.Y. 1967

10. Bergtraun E. M. - Op Cit pg 75 - 77

11. Priel V. Z. - Op Cit pg 197

CHAPTER SEVEN

MAINTAINING APPLICATION PACKAGES

7.0 INTRODUCTION

Unlike the inhouse-developed software, maintaining purchase software has its own unique problems. These include the contractual right to modify the purchased applications, the availability of source code and the DP staff's limited knowledge of the application packages.

In this chapter, two approaches to maintenance of application packages are discussed. These are (1):

1. maintaining through the computer

2. maintaining around the computer

The former involves changing the source code of the application packages while the latter uses preprocessor or postprocessor routines to meet the new requirement.

7.1 MAINTAINING THROUGH THE COMPUTER

Changing the purchased application source code usually involves contractual agreement with the software vendor. This type of maintenance forms an important part of the purchase/lease agreement. Perry (2) suggests the following steps should be undertaken in the signing of the maintenance contract:

Step 1. Develop an application packages maintenance policy. This involves:

1. deciding who shall perform the maintenance on the application package

2. evaluating the possibility of obtaining the source code from the vendor

3. formulating maintenance guidelines for the application package

The maintenance policy serves two important functions. Firstly, it is the basis for contractual negotiation with vendors. Secondly, it can be used as a guideline for the company DP staff in determining the types and extent of maintenance that can be performed on vendor software.

Step 2. Determine the maintenance proviso in the purchase contract. Major considerations in this step include :

1. Maintenance Responsibility

Responsibility for maintaining the purchased packages should be clearly defined in the contract. Consideration has to be given to ensuring that the maintenance does not only encompass error-correction but also user enhancements.

2. Quality of Maintenance Service

The quality of maintenance service should be mentioned in the contract. Considerations under this heading include:

- the response time for maintenance service
- the size, calibre and the location of the maintenance staff provided by the vendor

3. Quality of Documentation

Quality of documentation aids provided by vendor. This includes source code listing, system flowcharts, and transaction flow analysis.

4. The Cost of Maintenance Service

The contract should specify the cost of the maintenance service provided by the vendor.

5. The Location of the Service

Where the maintenance is to be carried out by the vendor should be stated in the contract. It is desirable for the maintenance to be done at company premises because it may provide learning opportunities for the company DP staff.

6. Penalty Clause

The penalty for inability or unwillingness to perform maintenance should also be included in the contract. Such clauses should be spelled out in terms of a daily penalty rate or a penalty for each occurrence.

7. Discontinuace of the Product

The contract should outline how future maintenance is to be handled in the event of the vendor going bankrupt or if the product is discontinued.

Step 3. Negotiate the contract

It is not uncommon for the vendor to disagree with the maintenance provisos proposed by the company. As such the company should attempt to arrange the provisos in order of importance so as to know which clauses of the provisions are expendable and which are essential.

Step 4. Monitor the contract

Once the purchase contract is signed, the company should monitor the maintenance proviso periodically as to its suitability to current needs. This periodic review also helps to inform new employees on the provisions of the contract.

7.2 MAINTENANCE AROUND THE COMPUTER

Often, modifying the application packages is a difficult task. This is because of the complexity of the packages and the lack of knowledge on the application package on the part of company DP staff.

In addition, with the frequent update of an application package by the vendor, the problem of having to repeat the maintenance on the new version of the software exists.

To a certain extent, maintenance on purchased application packages can be performed without having to modify the application source code.

The techniques in maintaining around the computer are (3):

1. manual processing

2. preprocessor

3. user exit facility

4. postprocessor

7.2.1 MANUAL PROCESSING

This involves implementing a manual processing system to cater for the new system requirement. For example, a manual log can be set up to record information that cannot be processed by the application package.

Note that manual processing is only a temporary maintenance solution.

7.2.2 PREPROCESSOR

Here, the data is processed by a preprocessor prior to entry into the application packages. Preprocessing is most valuable when the maintenance involves new editing of the input data, checking input to authorization files or analysing input data.

7.2.3 USER EXIT FACILITY

Alternatively, the company can use the user exit facilities provided by the application package. User exit facilities permit the company to incorporate a customer made program into the application during the execution of the application package. This type of change is useful when the company wishes to add additional reports or computation logic.

7.2.4 POSTPROCESSOR

Postprocessing involves inputing the output of the package into a postprocessing routine to achieve the desired output. For example, a series of output files from the package can be input to a program to produce a combined report.

In this chapter, two approaches to maintaining application packages are introduced. These maintenance approaches are not necessarily mutually exclusive but rather a supplementary.

REFERENCES

1. Perry W. E.
"Managing System Maintenance"
Q.E.D. Information Science Inc. 1981 pg 299 - 315

2. Perry W. E. - Op Cit pg 304 - 308

3. Perry W. E. - Op Cit pg 309 - 311

CHAPTER EIGHT

SUMMARY AND CONCLUSION

The objective of this report as stated in the beginning, is to provide an exploratory study of the concept of software maintenance management. The conclusions set out below are considered in the light of these aims.

SOFTWARE MAINTENANCE LIFE CYCLE

The traditional approach to the software life cycle is inadequate. Too little emphasis is placed on the maintenance activity which consume three-quarters of the resources. An expanded version of the life cycle consisting of a linear development phase and a cyclical maintenance phase is a more realistic software life cycle.

SOFTWARE MAINTENANCE PROBLEMS

Maintenance activities are affected by a host of problems. This report identifies these major problems to include user knowledge, programmer effectivenss, product quality, programmer availability and system volatility.

SOFTWARE MAINTENANCE CLASSIFICATION

All issues regarding maintenance are within the concept of "maintenance". Different opinions are put forward by prominent academics (Ogin, Mooney, Boehm, Lientz and Swanson). Analysis of these classifications led the writer to adopt of Lientz and Swanson's definition of maintenance (corrective, adaptive, preventive) in addition to support maintenance.

SOFTWARE MAINTENANCE FACTORS

Strategies for effective software maintenance cannot be formulated without thorough understanding of the factors affecting it. Two major categories of factors influencing the maintenance function have been identified in this report. The external factors (environmental variables) are staff quality, program quality, frequency and extent of maintenance and hardware volatality. The internal factors (program variables) include module independence, programming language, program style, program testability and program documentation.

FRAMEWORK FOR SOFTWARE MAINTENANCE MANAGEMENT

A major weakness of the traditional approach to software maintenance management is its ad hoc nature. To overcome this weakness, a systematic conceptual model of maintenance management is developed in this report. The main elements in this conceptual framework are:

1. Software Maintenance Objectives

The model starts with the identification of major software maintenance objectives. This includes software reliability, modifiability, maintainability and software efficiency.

2. Software Maintenance Strategy

Strategic planning forms part of the conceptual model. Here detailed steps in the development of a software maintenance strategy is discussed. The outcome of this strategic planning is a maintenance strategy which is consistent with the information strategy and ultimately the overall corporate strategy. The need for top level management involvement in these activities is also stressed.

3. Software Maintenance Information System

Information of an essential ingredient for effective maintenance decisions. To this end the model introduces the concept of a maintenance information system. This system not only provides relevant information to DP management but also helps to focus management attention on the importance of maintenance activities.

4. Software Maintenance Measurement Methodology

Maintenance effectiveness is difficult to measure because of the involvement of many exogenous factors (eg. user goodwill). The model suggests the use of an integrated approach consisting of subjective judgement approach and maintenance metrics to operationalize each facet of the maintenance activities, including software understandability, testability, modifiability, portability, reliability, usability and efficiency.

5. Software Maintenance Methods

With adequate understanding of the company's maintenance background, DP management is now in a position to develop software maintenance methods to deal with the maintenance function. The model indentifies two broad categories of maintenance methods; corrective methods and preventive methods. Major corrective methods considered are:

i. maintenance productivity tools

ii. maintenance quality assurance

iii. configuration managemant

iv. consultation programmer

v. maintainer education programme

vi. maintenance scheduling

vii. maintenance quality assurance programme

Preventive methods, up-front activities which begin once the software development is initiated, includeing:

i. fourth generation language

ii. user-driven computing

iii. rapid prototyping

iv. inclusion of maintainer in the development process

v. structured techniques

vi. use of DBMS interface

vii. documentation

viii. programming style

Each of the above methods together with its pitfalls are discussed.

To adopt the system approach, two additional issues are considered in this report. First the importance of the human aspect of software maintenance. Here the model suggests a maintainers' job enrichment programme to improve maintainers' morale and productivity.

Second, contractual issues in the maintenance of purchased application software are analysed. The model suggests two approaches to deal with this problem; "maintaining through the computer" and "maintaining around the computer".

The traditional approach to software maintenance management fails to recognize the complexity of systems maintenance and hence the necessity of having an integrated maintenance strategy such as that developed in this report.

The system approach presented in this report incorporates the

latest state of the art in software maintenance and recommends the use of currently available productivity tools.

With today´s rapidly increasing maintenance costs, DP management can no longer depend on the fire fighting approach to maintenance. Unless management can keep up with the latest developments in software maintenance, it is bound to be caught in the ever tightening squeeze between the demand for new applications and the pressures for maintaining the old ones.

BIBLIOGRAPHY

ALEX 68 — DATA MANAGEMENT
DEC 1968, PG 35-36
ALEXANDER G.
"CONTROL OF PROGRAM CHANGES"

ANDE 81 — AFIPS CONFERENCE PROCCEEDINGS
MAY 7 1981, PG 401-405
ANDERSON R. E.
"MODULAR DOCUMENTATION :A SOFTWARE DEVELPOMENT TOOL"

ANOF 72 — THE JOURNAL OF BUSINESS POLICY
No.4, 1972, PG 2-7
ANOFF H. I.
"THE CONCEPT OF STRATEGIC MANAGEMENT"

ANTH 65 — GRADUATE SCHOOL OF BUSINESS ADMINISTRATION,
HARVARD UNIVERSITY 1965
ANTHONY R. N.
"PLANNING AND CONTROL SYSTEMS : A FRAMEWORK FOR ANALYSIS"

ARNO 82 — IEEE TRANSACTIONS SOFTWARE ENGINNERING
1982, PG 10-27
ARNOLD R. S. AND PARKER D. A.
"THE DIMENTIONS OF HEALTY MAINTENANCE"

ARNO 84 — COMM. ACM, VOL 27, NO.11
NOV 1984, PG 1120-1121
ARNOLD R. S., SCHNEIDEWIND N. F. AND ZVEGINTZOV N.
"SOFTWARE MAINTENANCE WORKSHOP"

BASI 82 — IEEE TRANSACTIONS OF SOFTWARE ENGINERRING
SE-8 MAY 1982, PG 270-283
BASLI V. R. AND MILLS
"UNDERSTANDING AND DOCUMENTING PROGRAMS"

BASS 84 — AFIPS CONFERENCE PROCCEEDINGS
JULY 9-12 1984, PG 357-365
BASSETT P.
"SOFTWARE MANUFACTURING TECHNIQUES AND MAINTENANCE"

BELL 84 — AFIPS CONFERENCE PROCCEEDINGS
JULY 9-12 1984, PG 229-234
BELL F. J.

"TECHNOLOGY TRANSFER IN THE MAINTENANCE
ENVIROMENT"

BOEH 74 PROC. IFIP CONG.
1974, PG 192-197
BOEHM B. W.
"SOME STEPS TOWARD FORMAL AND AUTOMATIC AIDS TO
SOFTWARE REQUIREMENTS ANALYSIS AND DESIGN"

BOEH 75 IEEE TRANSACTIONS ON SOFTWARE ENGINEERING
SE-1, NO.1, MARCH 1975, PG 125-133
BOEHM B. AND OTHERS
"SOME EXPERIENCES WITH AUTOMATED AIDS TO THE
DESIGN OF LARGER SCALE RELIABLE SOFTWARE"

BOEH 78 "CHARACTERISTICS OF SOFTWARE QUALITY"
NEW YORK : TRW/NORTH-HOLLAND PUBLISHING CO
BOEHM B. AND OTHERS, 1978, PG 3.1-3.26

BOEH 81 "SOFTWARE ENGINEERING ECONOMICS"
ENGLEWOOD CLIFFS, NJ PRENTICE-HALL
BOEHM B. W., 1981, PG 35-51

BOEM 73 DATAMATION
MAY 1973, PG 48-59
BOEHM B. W.
"SOFWARE AND ITS IMPACT : A QUANTITIVE ASSESSMENT"

BRAN 75 IEEE COMPUTER SOCIETY CONFERENCE PROCEEDINGS
(COMP-CON)
SPRING 1975, PG 285-288
BRANTLEY C. L. AND OSAJIMA Y. R.
"CONTINUING DEVELOPMENT OF CENTRALLY DEVELOPED AND
MAINTAINED SOFTWARE SYSTEMS"

BRIC 83 AFIPS CONFERENCE PROCCEDINGS
MAY 16-19 1983, PG 113-121
BRICE L. AND CONNELL J.
"A METHODOLOGY FOR MINIMIZING MAINTENANCE COSTS"

BRIC 84 AFIPS CONFERENCE PROCCEEDINGS
JULY 9-12 1984, PG 209-216
BRICE L. AND CONNELL J.
"SYSTEM INFORMATION DATABASE :AN AUTOMATED
MAINTENANCE AID"

BRON 81 COMPUTERWORLD
JAN 1981
BRONSTEIN G. M. AND OKAMOTO R. I.
"I´M OK ,YOU´RE OK MAINTENANCE IS OK "

BUDL 84 AFIPS CONFERENCE PROCCEDDINGS
JULY 9-12 1984, PG 389-394
BUDLONG F. C.

"COMMERICAL AND MILITARY SOFTWARE DOCUMENTATION
:DIFFERENT STEPS TO COMMON GOALS"

CANN 79 EDP ANALYSER
OCT 1979, PG 1-4
CANNING R.
"THE MAINTENANCE ICEBERG"

CANN 72 EDP ANALYSER
10, 7 JULY 1972, PG 1-14
CANNING R. G.
"MODULAR COBOL PROGRAMMING"

CARN 78 COMPUTERS AND PEOPLE
JUNE 1978, PG 17-27
CARNEGIE, DALE
"SEMI-CODE IN DESIGN AND MAINTENANCE"

CAVE 78 IEEE TRANSACTIONS OF SOFTWARE ENGINERRING
SE-4 JULY 1978, PG 326-334
CAVE W. C. AND SALIBURY A. B.
"CONTROLLING THE SOFTWARE LIFE CYCLE : THE PROJECT
MANAGEMENT TASK"

CENT 82 AFIPS CONFERENCE PROCCEEDINGS
JUNE 7-10 1982, PG 399-407
CENTRE J. W.
"A QUALITY ASSURANCE PROGRAM FOR SOFTWARE
MAINTENANCE"

CHAM 78 DATAMATION
SEPT 1978, PG 105-206
CHAMPINE G. A.
"WHAT MAKE A SYSTEM RELIABLE"

CHAM 83 SOFTWARE ENGINEERING NOTES, VOL 8, NO.3
JULY 1983
CHAMBERS J.
"EVOLUTION AND MAINTENANCE ARE BOTH SEMANTICALLY
UNSOUND AND ALSO HAZARDOUS TO THE HEALTH OF OUR
PUBLIC IMAGE"

CHAP 81 AFIPS CONFERENCE PROCCEEDINGS
MAY 7 1981, PG 345-352
CHAPIN N.
"PRODUCTIVITY IN SOFTWARE MAINTENANCE "

COCH 83 COMPUTERWORLD
VOL 17, 1983, PG 47-49
COCHAM H. T.
"FOURTH GENERATION LANGUAGES"

CONN 83 AFIPS CONFERENCE PROCEEDINGS
MAY 1983, PG 113-121
CONNELL J. AND BRICE L.

"A METHODOLOGY FOR MINIMIZING MAINTENANCE COSTS"

CONN 84 AFPIS CONFERENCE PROCCEEDINGS
JULY 9-12 1984, PG 243-249
CONNELL J. AND BRICE L.
"PROLONGING THE LIFE OF SOFTWARE "

COOP 78 IEEE TRANSACTIONS OF SOFTWARE ENGINERRING
SE-4, JULY 1978, PG 319-325
COOPER J. D.
"CORPORATE LEVEL SOFTWARE ENGINERRING"

CULP 75 IEEE TRANSACTIONS ON SOFTWARE ENGINEERING
SE1(2), 1975, PG 174-178
CULPEPPER L. M.
"A SYSTEM FOR RELIABLE ENGINEERING SOFTWARE"

CURT 79 IEEE TRANSACTIONS OF SOFTWARE ENGINERRING
SE-5, MAR 1979, PG 96-104
CURTIS C. AND OTHERS
"MEASURING THE PSYCHOLOGICAL COMPLEXITY OF
SOFTWARE MAINTENANCE TASK WITH THE HALSTEAD AND
McCABE METRICS"

DALY 77 IEEE TRANSACTIONS OF SOFTWARE ENGINERRING
SE-3, MAY 1977, PG 227-242
DALY E. B.
"MANAGEMENT OF SOFTWARE DEVELOPMENT "

DEBA 75 COMPUTER
JUNE 1975, PG 64-65
DE BALBINE GUY
"THE STRUCTURING ENGINE: A TRANSITIONAL TOOL"

DENN 81 COMMUNICATIONS OF ACM
VOL 24, FEB 1981, PG 57-58
DENNING
"THROWAWAY PROGRAMS"

DIJK 68 COMMUNICATIONS OF THE ACM
VOL 11, NO 3, MARCH 1968
DIJKSTRA E.
"GOTO STATEMENT CONSIDERED HARMFUL"

DONA 73 DATAMATION
DEC 1973, PG 52-54
DONALDSON, JAMES R.
"STRUCTURED PROGRAMMING"

EJIO 83 SOFTWARE ENGINEERING NOTES, VOL 8, NO.2
APR 1983
EJIOGU L. O.
"EVOLUTION IS SIGNIFICANTLY SUPERFLUOUS;
MAINTENANCE IS SEMANTICALLY SOUND"

ELEH 82 COMMUNICATIONS OF ACM
VOL 25, AUG 1982, PG 512-521
ELSHOFF J. L. AND MARCOTTY M.
"IMPROVING COMPUTER PROGRAM READABILITY TO AID MAINTENANCE"

ELLI 75 DATA MANAGEMENT
NOV 1975, PG 32-35
ELLIOT R. W. AND STORY E. R.
"AUDIO TECHNIQUES FOR PROGRAM DOCUMENTATION"

ELSH 76 ACM SIGPLAN NOTICES
MAY 1976, PG 38-46
ELSHOFF J.
"MEASURING COMMERCIAL PL/1 PROGRAMS USING HALSTEAD'S CRITERIA"

ELSH 82 COMMUNICATIONS OF THE ACM
AUGUST 1982, PG 512-521
ELSHOFF J. L. AND MARCOTTY M.
"IMPROVING COMPUTER PROGRAM READIBILITY TO AID MODIFICATION"

FELD 79 SOFTWARE PRACTICE AND EXPERIENCE
VOL 9, # 4, APRIL 1979, PG 225-265
FELDMAN S. I.
"MAKE - A PROGRAM FOR MAINTAINING COMPUTER PROGRAMS"

FEUE 79 AFIPS CONFERENCE PROCCEEDINGS
JUNE 4-7 1979, PG 1003-1012
FEUER A. R. AND FOWLERS BE. B.
"RELATING COMPUTER PROGRAM MAINTAINABILITY TO SOFTWARE MEASURES"

FINK 77 COMPSAC 77
1977, PG 533-538
FINK R. C.
"MAJOR ISSUES INVOLVING THE DEVELOPMENT OF ALL EFFECTIVE MANAGEMENT CONTROL SYSTEMS FOR SOFTWARE MAINTENANCE"

FRAS 80 SOFTWARE PRACTICE AND EXPERIENCE
VOL 10, #10, OCT 1980, PG 817-822
FRAZE C.
"MAINTAINING PROGRAM VARIANT BY MERGING EDITOR SCRIPTS"

GANE 79 "STRUCTURED SYSTEMS ANALYSIS: TOOLS AND TECHNIQUES"
PRENTICE-HALL, ENGLEWOOD NJ
GANE, CHRIS AND SARSON, 1979

GILB 77 "SOFTWARE METRICS"
CAMBRIDGE, MASSACHUSETTS: WINTHROP PUBLISHERS
GILB TOM, 1977

GILB 77 "SOFTWARE METRICS"
WINTHROP PUBLISHERS INC CAMBRIDGE MA
GILB T., 1977, PG 26-49

GILB 77 COMPUTERS AND PEOPLE
SEPT 1977, PG 16-21
GILB TOM
"THE MEASUREMENT OF SOFTWARE RELIABILITY AND MAINTAINABILITY: SOME UNCONVENTIONAL APPROACHES TO RELIABLE SOFTWARE"

GILB 77 COMPUTER WEEKLY
SEPT 22, 1977
GILB TOM
"GUARANTEEING PROGRAM MAINTAINABILITY (GILB'S MYTHODOLOGY)"

GILB 78 COMPUTER WEEKLY
JAN 15, 1978
GILB TOM
"WHY STRUCTURED PROGRAMS DON'T ALWAYS EASE MAINTENANCE (GILB'S MYTHODOLOGY)"

GILB 79 COMPUTER WEEKLY
MARCH 15, 1979
GILB TOM
"NEED FOR TRAINING IN PROGRAM MAINTENANCE (GILB'S MYTHODOLOGY)"

GLAD 82 SOFTWARE ENGINEERING NOTES
7, 1982, PG 35-39
GLADDEN G. R.
"STOP THE LIFE CYCLE, I WANT TO GET OFF"

GLAS 79 "SOFTWARE RELIABILITY GUIDEBOOK"
PRENTICE-HALL
GLASS R. L., 1979

GLAS 81 "SOFTWARE MAINTENANCE GUIDEBOOK"
PRENTICE-HALL INC NJ
GLASS R. L. AND NOISEUX R. A., 1981, PG 33

GROS 84 COMPUTER, VOL 17, NO.1
JAN 1984, PG 88
GROSBERG J. A.
"REPLACING THE TERM 'SOFTWARE MAINTENANCE'"

GUIM 83 COMMICATIONS OF ACM
OCT 1983, VOL 26, PG 739-746
GUIMARAES T.

"MANAGING APPLICATION PROGRAM MAINTENANCE
EXPENDITURES"

GUND 73 DATAMATION
VOL 19, JUNE 1973, PG 99-101
GUNDER R. E.
" A GLIMPSE INTO PROGRAM MAINTENANCE"

HALS 77 "ELEMENTS OF SOFTWARE SCIENCE"
NORTH-HOLLAND ELSEVIER, NY
HALSTEAD H. M., 1977, PG 84-91

HARR 83 JOURNAL OF SYSTEMS MANAGEMENT
JULY 1983, PG 10-14
HARRISON W., KENNETH M. AND KLUCZNV R.
"RESEARCH IN SOFTWARE MAINTENANCE"

HARR 83 JOURNAL OF SYSTEMS MANAGEMENT (USA)
JULY 1983, PG 10
HARRISON W. AND OTHERS
"RESEARCH IN SOFTWARE MAINTENANCE"

ISAA 80 COMPUTER, VOL 13,1
JAN 1980, PG 4-6
ISAACSON P. AND JULIUSSEN E.
"GUEST EDITORIAL : WINDOW ON THE 80'S"

JACK 75 "PRINCIPLES OF PROGRAM DESIGN"
ACADEMIC PRESS
JACKSON M. A., 1975

JONE 78 COMMUNICATIONS OF ACM
VOL 21, 1978, PG 882
JONES R. A.
"MAINTENANCE CONSIDER HARMFUL"

JONE 78 COMPUTER-WORLD
JAN 23, 1978, PG 25,30
JONES, ROBERT R.
"CREATIVITY SEEN VITAL FACTOR EVEN IN MAINTENANCE
WORK"

JONE 78 IBM SYSTEMS JOURNAL
VOL 17, NO.1, 1978, PG 39-63
JONES, CAPERS
"MEASURING PROGRAMMING QUALITY AND PRODUCTIVITY"

KERN 84 "THE ELEMENTS OF PROGRAMMING STYLE"
MCGRAW-HILL N.Y.
KERNIGAN B. W. AND PLAUGER P. J., 1984

KHAN 75 CANADIAN DATA SYSTEMS
MARCH 1975, PG 30-32
KHAN Z.

"HOW TO TACKLE THE SYSTEMS MAINTENANCE DILEMMA"

LEHM 80 PROC IEEE
1980, PG 1060-1076
LEHMAN M. M.
"PROGRAM, LIFE CYCLES, AND THE LAWS OF SOFTWARE EVOLUTION"

LEHM 83 SOFTWARE ENGINEERING NOTES, VOL 8, NO.5
OCT 1983, PG 39
LEHMAN M. M.
"MORE ON EVOLUTION"

LIEN 78 COMMUNICATIONS OF ACM
VOL 27, 1978, PG 466-471
LIENTZ B. P. AND SWANSON G. E. AND TOMKINS G. E.
"CHARCTERISTICS OF APPLICATION SOFTWARE MAINTENANCE"

LIEN 78 DATA MANAGEMENT
VOL 16, SEPT 1978, PG 15-18
LIENTZ B.P. AND SWANSON E. B.
"DISCOVERING ISSUES IN APPLICATION SOFTWARE MAINTENANCE "

LIEN 79 DATA MANAGEMENT
VOL 17, APRIL 1979, PG 26-30
LIENTZ B. P. AND SWANSON E. B.
"SOFTWARE MAINTENANCE : A USER/MANAGEMENT TUG-OF-WAR"

LIEN 80 "SOFTWARE MAINTENANCE MANAGEMENT"
ADDISON-WESLEY
LIENTZ B. P. AND SWANSON E. B. 1980 PG 115-140

LIEN 81 COMMUNICATIONS OF ACM
VOL 24, NOV 1981, PG 763-769
LIENTZ B. P. AND SWANSON G. E.
"PROBLEMS IN APPLICATION SOFTWARE MAINTENANCE"

LIEN 84 COMPUTING SURVEYS
VOL 15, # 3, SEPT 1983, PG 271-278
LIENTZ B. P.
"ISSUES IN SOFTWARE MAINTENANCE "

LIND 73 DATAMATION
VOL 19, MAY 1973, PG 64-71
LINDHORST W. M.
"SCHEDULED MAINTENANCE OF APPLICATIONS SOFTWARE"

LIUC 76 DATAMATION
VOL 22, NOV 1976, PG 51-55
LIU C. C.
"A LOOK AT SOFTWARE MAINTENANCE"

LUND 78 "INFORMATION SYSTEM DEVELOPMENT - A FIRST INTRODUCTION A SYSTEMATIC APPROACH"

DEPARTMENT OF INFORMATION PROCESSING COMPUTER SCIENCE, UNIVERSITY OF STOCKHOLM
LUNDEBERG M., GOLDKUHL G. AND NILSSON A., 1978

LYON 81 AFIPS CONFERENCE PROCCEEDINGS
MAY 7 1981, PG 337-341
LYONS M. J.
" SALAVING YOUR SOFTWARE ASSETS (TOOLS BASED MAINTENANCE) "

MANC 82 COMPUTER MANAGEMENT
MAR 1982, PG 26
MANCHESTER J.
" PROGRAM MAINTENANCE"

MARS 83 AFIPS CONFERENCE PROCCEEDINGS
MAY 16-19 1983, PG 145-153
MARSH R. E.
"APPLICATION MAINTENANCE : ONE SHOP'S EXPERENCE AND ORGANIZATION "

MARS 83 AFIPS CONFERENCE PROCCEEDINGS
MAY 16-19 1983, PG 131-136
MARSELOS N. L.
"HUMAN INVESTMENT TECHNIQUES FOR EFFECTINE SOFTWARE MANAGEMENT "

MART 83 "SOFTWARE MAINTENANCE : THE PROBLEMS AND ITS SOLUTIONS"
ENGLEWOOD CLIFFS PRENTICE-HALL
MARTIN J. AND McCLURE C. 1983 PG 453 - 465

MART 83 "SOFTWARE MAINTENANCE : THE PROBLEMS AND ITS SOLUTIONS"
PRENTICE-HALL INC NJ
MARTIN J. AND McCLURE C., 1983, PG 43

MATH 84 AFIPS CONFERENCE PROCCEEDINGS
JULY 9-12 1984, PG 395-403
MATHER D.
"ONE PERSON'S PERCEPTION OF MILITARY DOCUMENTATION"

MCCA 76 IEEE TRANS ON SOFTWARE ENGINEERING
VOL SE-2, No.4, DEC 1976, PG 308-320
McCABE T.
"A COMPLEXITY MEASURE"

MCCL 78 "REDUCING COBOL COMPLEXITY THROUGH STRUCTURED PROGRAMMING"
VAN NOSTRAND REINHOLD CO NY
McCLURE C. 1978, PG 77 - 121

MCCO 84 AFIPS CONFERENCE PROCCEEDINGS
JULY 9-12 1984, PG 273-281

McCONNELL P. R. H. AND WOLFGANG B. S.
"RESULTS OF MORDEN SOFTWARE ENGINERRING PRINCIPLES
APPLIED TO SMALL AND BIG PROJECTS "

MCCR 73 DATAMATION
DEC 1973, PG 50-51
MC CRACKEN D.D.
"REVOLUTION IN PROGRAMMING: AN OVERVIEW"

MILL 76 DATAMATION
DEC 1976, PG 67-70
MILLER, LARRY K. AND DAVID OSTROM
"SOURCE PROGRAM MANAGEMENT"

MOON 75 DATAMATION
VOL 21, FEB 1975, PG 63-66
MOONEY J. W.
"ORGANIZATION PROGRAM MAINTENANCE"

MYER 76 "SOFTWARE RELIABILITY"
JOHN WILEY N. Y.
MYERS G.J., 1976

McGR 72 DATA PROCCESSING
MAY-JUNE 1973, PG 172
McGREGOR B.
"PROGRAM MAINTENANCE"

McKE 84 AFIPS CONFERENCE PROCCEEDINGS
JULY 9-12 1984, PG 187-193
McKEE J. R.
"MAINTENANCE AS FUNCTION OF DESIGN"

NARR 84 AFIPS CONFERENCE PROCCEEDINGS
JULY 9-12 1984, PG 235-245
NARROW B. AND KELLY I.
"TWO PERCEPTIONS OF SOFTWARE MAINTENANCE PERFORMED
BY ON-SITE CONTRACTOR"

NOLA 79 HARVARD BUSINESS REVIEW
MAR-APRIL 1979, PG 115-126
NOLAN R. L.
"MANAGING THE CRISES IN DATA PROCESSING"

NONE 72 DATA PROCCESSING
SEPT-OCT 1973, PG 314-317
"PROGRAM MAINTENANCE :USER VIEW "

OGDI 72 DATAMATION
JULY 1972, PG 71-72, 75- 78
OGDIN, JERRY L.
"DESIGNING RELIABLE SOFTWARE"

OGDI 77 DATAMATION
VOL 18 JULY 1977, PG 71-78

OGDINOG J. L.
"DESIGNING RELIABLE SOFTWARE "

PARI 82 "TECHNIQUES OF PROGRAM AND SYSTEM MAINTENANCE"
WINTHROP PUBLISHERS INC
PARIKH G., 1982, PG 53-55

PARI 84 SOFTWARE ENGINEERING NOTES, VOL 9, NO.2
APR 1984, PG 114
PARIKH G.
"WHAT IS SOFTWARE MAINTENANCE REALLY? WHAT IS IN A NAME?"

PARK 82 DATA COMMUNICATION
VOL 11, # 1, JAN 1982, PG 93-100
PARKS M.
" BOOST PROGRAMMER PRODUCTITY WITH SOFTWARE TOOLS "

PARN 72 COMMUNICATIONS OF THE ACM
DEC 1972, PG 1053-1058
PARNAS D.L.
"ON THE CRITERIA TO BE USED IN DECOMPOSING SYSTEMS INTO MODULES"

PARN 79 IEEE TRANSACTIONS OF SOFTWARE ENGINEERING
SE 5, 2, MARCH 1979, PG 128-137
PARNES D.
"DESIGNING FOR EASE OF EXTENSION AND CONTRACTION"

PERR 81 "MANAGING SYSTEMS MAINTENANCE"
QED INFORMATION SCIENCES INC
PERRY W. E., 1981, PG 287-288

PETE 78 ACM PROCEEDINGS OF SOFTWARE QUALITY AND ASSURANCE WORKSHOPS
3, 1978, PG 67-71
PETERS L.
"RELATING SOFTWARE REQUIREMENTS AND DESIGN"

POME 72 IBM SYSTEMS JOURNAL
VOL 11, NO.3, 1972, PG 234-254
POMEROY J.W.
"A GUIDE TO PROGRAMMING TOOLS AND TECHNIQUES"

PUNT 75 DATA PROCCESSING
SEPT-OCT 1975, PG 292-294
PUNTER M.
"PROGRAMMING FOR MAINTENANCE"

RANY 83 AFIPS CONFEREMCE PROCCEEDINGS
MAY 16-19 1983, PG 173-180
RANYOR R. I. AND SPECKMANN L. D.

"MAINTAINING USER PARTICIPATION THOUGHOUT THE SYSTEMS DEVELOPMENT CYCLE"

REUT 81 AFIPS CONFERENCE PROCCEEDINGS
MAY 7 1981, PG 343-347
REUTTER J.
"MAINTENANCE IS A MANAGEMENT PROBLEM AND PROGRAMMER OPPORTUNITY "

RICH 83 AFIPS CONFERENCE PROCCEEDINGS
MAY 16-19 1983
RICHARDSON G. Y. AND BULTER W. C.
"ORGANIZATIONAL ISSUES ON EFFECTINE MAINTENANCE MANAGEMENT "

RICH 84 AFIPS CONFERENCE PROCCEEDINGS
JULY 9-12 1984, PG 203-208
RICHARDSON AND HOLID E. D.
"REDOCUMENTATION : AN ADDRESSING TO MAINTENANCE LEGACY"

RIGG 69 DATAMATION
NOV 1969, PG 227-235
RIGGS R.
" COMPUTER SYSTEM MAINTENANCE"

SCHN 83 AFIPS CONFERENCE PROCCEEDINGS
MAY 16-19 1983, PG 137-144
SCHNEIDER G. R. E.
"STRUCTURED SOFTWARE MAINTENANCE "

SELL 84 AFIPS CONFERENCE PROCCEEDINGS
JULY 9-12 1984, PG 195-202
SELLER A. M.
"MAINTAINING USER SATISFACTION WITH PERFORMANCE ON LINE SYSTEM"

SHAR 77 PROCCEEDINGS COMPSAC 77
1977, PG 520-526
SHARPLEY W. K.
"SOFTWARE MAINTENANCE PLANNING FOE EMBEDDED COMPUTER SYSTEMS"

SHNE 80 "SOFTWARE PSYCHOLOGY : HUMAN FACTORS IN COMPUTER AND INFORMATION SYSTEM"
WINTHROP PUBLISHERS INC CAMBRIDGE MA
SHNEIDERMAN B. 1980, PG 93-122

SHNI 80 "SOFTWARE PSYCHOLOGY"
CAMBRIDGE WA, WINTHROP PUBLISHERS INC
SHNIEDERMAN B., 1980

SOMM 82 "SOFTWARE ENGINEERING"
INTERNATIONAL COMPUTER SCIENCE SERIES ADDISON-WESLEY PUBLISHING CO

SOMMERVILLE I., 1982, PG 198-203

STAM 83 MIS WEEK
4, 1983, PG 18-19
STAMPS D.
"HAIL 4TH GENERATION LANGUAGES"

STEV 74 IBM SYSTEMS JOURNAL
VOL 13, No.2, 1974, PG 115-139
STEVENS W. AND OTHERS
"STRUCTURED DESIGN"

SWAN 76 IEEE 2ND INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING PROCEEDINGS
OCT 1976, PG 492-497
SWANSON E. B.
"THE DISCUSSIONS OF MAINTENANCE"

TALI 71 SOFTWARE PRACTICE AND EXPERIENCE
VOL 1, #3, JULY-SEPT 1971, PG 254-258
TALIAFERRO W. M.
"MODULARITY : THE KEY TO SYSTEM GROWTH POTENTIAL"

TASS 78 "PROGRAM STYLE, DESIGN, EFFICIENCY, DEBUGGING AND TESTING"
PRENTICE-HALL INC NJ
TASSEL D. V., 1978, PG 238-284

TATE 85 NZCS PAPERS
1985 PG 325 - 340
TATE G. AND HAYWARD R. J.
"THE BRAVE NEW WORLD OF NGLS LESSONS FROM DP HISTORY"

TAUT 83 AFIPS CONFERENCE PROCCEEDINGS
MAY 16-19 1983, PG 123-129
TAUTE B. J.
"QUALITY ASSURANCE AND MAINTENANCE APPLICATION SYSTEMS."

THAY 78 "SOFTWARE RELIABILITY"
NORTH HOLLAND AMSTERDAM
THAYER T. A., LIPOW M AND NELSON E. C., 1978

TINN 83 AFIPS CONFERENCE PROCCEEDINGS
MAY 16-19 1983, PG 107-112
TINNIRELLO P. C.
" IMPROVING SOFTWARE MAINTENANCE ATTITUDES"

TINN 84 AFIPS CONFERENCE PROCCEEDINGS
JULY 9-12 1984, PG 251-257
TINNRELLO P. C.
"SOFTWARE MAINTENANCE IN FOURTH GENERATION ENVIRONMENTS"

VAN 78 "PROGRAM STYLE, DESIGN, EFFICIENCY,DEBUGGING AND TESTING"
PRENTICE-HALL, NJ
VAN TASSEL D., 1978, PG 183-187

VESS 83 COMMUNICATIONS OF ACM
VOL 26, FEB 1983, PG 128-134
VESSEY I. AND WEDER R.
"SOME FACTORS AFFECTING PROGRAM REPAIR MAINTENANCE :AN EMPIRICAIL STUDY"

WALK 78 DATAMATION
SEPT 1978, PG 211-214
WALKER M. G.
"A THEORY FOR SOFTWARE RELIABILITY"

WALS 77 IBM SYSTEMS JOURNAL
VOL 16, No.1, 1977, PG 54-75
WALSTON L. E. AND FELIX C. P.
"A METHOD OF PROGRAM MEASUREMENT AND ESTIMATION"

WEGN 78 3RD INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING PROC
MAY 10-12, 1978, PG 243-259
WEGNER P.
"RESEARCH DIRECTIONS IN SOFTWARE TECHNOLOGY"

WEGN 78 IEEE 3RD INTERNATIONAL CONFERENCE ON SOFTWARE PROC.
MAY 1978, PG 243-249
WEGNER P.
"RESEARCH DIRECTIONS IN SOFTWARE TECHNOLOGY"

WEIN 80 "WORST-FIRST MAINTENANCE IN TECHNIQUES OF PROGRAM AND SYSTEM MAINTENANCE"
LINCOLN N. E. : ETHNOTECH
WEINBERG G, 1980, PG 119-121

WELS 82 COMMUNICATIONS OF ACM
VOL 25, JULY 1982, PG 446-452
WELSER M.
"PROGRAMMER USES SLICE WHEN DEBUGGING"

WIRT 71 COMMUNICATIONS OF ACM
VOL 14, NO.4, APRIL 1971, PG 403-415
WIRTH N.
"PROGRAM DEVELOPMENT BY STEPWISE REFINEMENT"

YAUS 78 IEEE 2ND INTERNATIONAL COMPUTER SOFTWARE AND APPLICATION CONFERENCE PROC.
YAU S., COLLONFELLO J. AND McGREGOR T.
"RIPPLE EFFECT ANALYSIS ON SOFTWARE MAINTENANCE"

YAUS 78 PROCCEEDINGS COMPSAC 78
1978, PG 60-65

YAU S. S. AND COLLOFELLO J. S. AND MACGREGOR T.
"RIPPLE EFFECT ANALYSIS OF SOFTWARE MAINTENANCE"

YAUS 79 PROCCEEDINGS COMPSAC 79
YAU S. S. AND COLLOFELLO J. S.
1979, PG 679-697
"SOME STABILITY MEASURES FOR SOFTWARE MAINTENANCE"

YAUS 80 IEEE TRANSACTIONS OF SOFTWARE ENGINERRING
SE-6 NOV, 1980, PG 545-552
YAU S. S. AND COLLOGENLLO J.
"SOME STABILITY MEASURES FOR SOFTWARE
MAINTENANCE"

ZELK 79 "PRINCIPLES OF SOFTWARE ENGINEERING AND DESIGN"
PRENTICE-HALL, NJ
ZELKOWITZ M., 1979, PG 2-11

ZVEG 81 COMPUTERWORLD
MAR 1981
ZVEGINTZOV N.
"TIPS TO BOOST MAINTENANCE PROGRAMMER MORALE"

ZVEG 82 AFPIS CONFERENCE PROCCEEDINGS
JUNE 7-10 1982, PG 561-586
ZVEGINTZOV N.
"WHAT LIFE ? WHAT CYCLE? "