Information Systems Maintenance: An Integrated Perspective

By: Chris Edwards
Professor of Management
Information Systems
Cranfield School of Management
Cranfield, England

Abstract

This article argues that information systems maintenance is a more complex and integrated task than portrayed in the literature. It involves not only the maintenance of the applications software, but also all the other elements in an operational system. The literature relating to the maintenance of each element is reviewed to reveal substantial underdevelopment in some areas and fragmentation between elements. The present practice of focusing upon the maintenance of particular individual elements is criticized and a new focus upon changes in the information inputs to the systems development process is proposed. Alternative methods of managing the maintenance operation are examined and the implications of these methods in terms of designing the procedures, staffing the maintenance function, and the need for communication are discussed.

Keywords: Information systems maintenance; systems life cycle; systems life cycle management.

ACM Categories: K.6.1

Introduction

A great deal of academic and practical attention has been devoted to studying methods of developing computer-based information systems; considerably less attention has been devoted to studying the management of systems once implemented. The latter task is known variously as systems maintenance, applications maintenance, systems management, systems audit, post-implementation management, or applications management.

Traditionally, the creation of information systems has been discussed by considering each task in a systems development cycle. In essence, that is a list of tasks required to be undertaken in the process of creating an information system. The development phase begins the systems life cycle which traces the life of a system from inception to replacement. The latter part of the life cycle, namely systems operation, is usually discussed by operational task with little reference to any particular system. For instance, installation, security, file management, and engineering maintenance are usually discussed without reference to any particular operational system. Figure 1 shows the matrix of operational tasks versus systems, highlighting the conventional discussion perspective.

Not only is discussion organized in this way, but personnel are similarly allocated; analysts and programmers to system and operations personnel to tasks. Except in the largest installations offering very great development specialization, systems creation involves very few functional specializations whereas subsequent operation usually involves a greater number. Personnel involved in operations may work on a number of systems in one day, whereas development personnel may work for many months on a single system. For this reason, operation demands a greater degree of functional integration. This article is concerned with the process of integration of elements of operational systems: a global view of related post-implementation tasks is proposed which emphasizes the systems rather than the functional perspective.

The very early development of commercial systems tended to emphasize the problems of programming, since programming was very new and the business task to be developed was usual-
Systems Maintenance

Definitions of maintenance vary in a computer-related context. Boehm [8] distinguishes between software update and software repair. The former involves enhancements whereas the latter ensures that the programs perform as first intended. Oglin [27] includes both of these under maintenance but introduces the term modification to include changes that result from a developing environment. Riggs [20] moves toward a wider view of maintenance in his definition:

"Systems maintenance is 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 government and other agencies."

Swanson's [30] highly developed classification of the demands for software maintenance consists of:

A. Corrective Maintenance arising from:
   1. Processing failure to produce data.
   2. Performance failure to meet specified requirements.
   3. Implementation failure that has not yet led to 1 or 2 above.

B. Adaptive maintenance arising from:
   1. Changes in the data environment.
   2. Changes in the processing environment
C2

The System

Application Software

A1 A2, A3, C1, C3.

Hardware and Systems Software

B2

Environment

Figure 2 — A Representation of Swanson's Demands for Software Maintenance

involving hardware or systems software changes necessitating amendments to applications software.

C. Perfective maintenance arising from:

1. Processing inefficiency of software which meets processing specifications yet can be improved.
2. Performance enhancement by amending, or adding reports, or facilities.
3. Maintainability improvements to make programs more easily maintainable.

This classification can be represented in the form of Figure 2.

The system is seen to consist of applications software and hardware, and to operate within a systems environment. Each of Swanson's demands for maintenance is located in its originating systems element and cross referenced with the earlier list. The arrows show that wherever a demand arises the effect is to cause an amendment to the applications software. Changes in the data environment (B1) presumably occur either because of changes in some other unspecified system element or in the organizational environment itself.

Systems maintenance, as portrayed in this article, is considerably wider in scope than Swanson's view of software maintenance. Figure 3 shows the operational system in the same format as Figure 2, but consisting of rather more elements than Swanson's included.

Any element may require modification either because of an embodied existing error in that element (X), some change occurring in some other element (Y), or some change occurring in the environment (Z). A change in any element may demand a consequent change in itself (U), in some other element (V), or in the environment (W). Changes in one element can develop whole
Figure 3 — A Representation of Demands for Systems Maintenance
Information Systems Maintenance

chains of required amendments involving numerous systems elements.

Building upon this, systems maintenance is defined here as "an alteration to any systems element necessary to eliminate a fault in that element, adapt to a change in that or any other element, or to improve upon the performance characteristics of an element." In essence, this is merely an expansion and generalization of Swanson's classification to allow for other system elements. For example, A1, B2, and C2 in Figure 2 are particular examples of X, Y and Z in Figure 3, respectively. However, this expansion in the scope of the definition changes the whole nature of the maintenance task and, in particular, demands a reconsideration of how the expanded requirements might be operationalized.

Systems Life Cycle

The development life cycle has been defined by many authors with only minor variations. Alternative task titles and the level of detail envisaged by the individual authors make differences appear more significant than is actually the case. In essence, a development life cycle is a structured method of producing the elements required for a system. For the purposes of this article, the life cycle is to consist of the tasks shown in Figure 4. This life cycle consists of 18 tasks which can be grouped into four stages:

**Systems Overview** — Tasks 1 and 2 involve investigating the nature of the problem and undertaking a brief appraisal as to its suitability for computer processing.

**Systems Design** — Tasks 3 through 6 demand a detailed consideration of user requirements and the proposal of a number of outline solutions. The last task in this stage involves evaluating the alternatives and selecting an appropriate action plan.

**Systems Creation** — Tasks 7 through 15 are the core of the development process and involve a great deal of effort. This is the stage in which the majority of the elements required to utilize the system are created.

**Systems Implementation** — Tasks 16 through 18 focus on implementing the organizational changes necessary to utilize the newly developed system.

Each task can be classified either as a control task or as a production task. Control tasks direct the flow through the development cycle whereas production tasks develop elements of the required system. For example, task 1 is an investigation which results in a report which is used in task 2. Hence both task 1 and 2 might be viewed as control tasks as together they direct the subsequent flow. Contrast this with task 10 which involves the production of software—an element in the operational system. Some tasks result in a statement of requirements which are then embodied in a number of other systems elements. Such tasks are still productions tasks. The devising of security specifications would be an example of such a task as the resulting controls are embodied in the software, the manual systems, and possibly other elements. Usually a production task will result in some form of documentation that will survive for the entire life of the system and be updated as changes occur.

Figure 5 recasts the development life cycle labeling each task as production (p) or control (c), and shows the resulting systems elements. Interactions between tasks and systems elements are indicated.

An arrow pointing toward a systems element (i.e., upwards) indicates that the task in that row results in the systems element in that column. An arrow pointing toward a task (i.e., to the left) indicates that information from the element in that column is used in the task. For example, the task of security design results in a security specification which is then used in the tasks of computer systems design and manual systems design. Systems elements of hardware and systems software are not included, as these are essentially installation specific and are system dependent to a very limited extent. The figure does not attempt to portray iteration within tasks and between tasks as this would confuse the situation.

Most of the systems development tasks draw information from the organization and its environment. Figure 6 shows the relevant organizational information added to Figure 5. An arrow pointing towards a particular task portrays that the type of information specified by that column is used in that task. For example, to adequately analyze...
user needs for information would require data relating to the business environment, the business organization, characteristics of users, and details of the task to be undertaken. Precise details of the information required is system- and organization-dependent. The development cycle can thus be seen as a structured set of procedures for gathering information relating to the organization and transforming this into the elements required for an operational system. Task 18, namely the post-implementation audit, is essentially a non-repetitive activity and hence information inputs for this task do not need to be monitored other than immediately following implementation.

Each of the resulting systems elements has a time dimension and may or may not need to be updated as the original information inputs to that element change. Figure 7 is a cross-reference of information inputs and the resulting systems elements. This has been compiled from Figure 6 by tracing systems elements through the development cycle to the information required for each element. When information directly relates to an element it is shown in Figure 7 by an upward pointing arrow. The dots represent situations in
<table>
<thead>
<tr>
<th>TASKS</th>
<th>Production (P) or Control (C)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Post-implementation report</td>
</tr>
<tr>
<td></td>
<td>Report on systems testing</td>
</tr>
<tr>
<td></td>
<td>Cost/Value justification</td>
</tr>
<tr>
<td></td>
<td>Feasibility report</td>
</tr>
<tr>
<td></td>
<td>Specification of user needs</td>
</tr>
<tr>
<td></td>
<td>Systems specification (overview)</td>
</tr>
<tr>
<td></td>
<td>Cost/Value justification</td>
</tr>
<tr>
<td></td>
<td>Security specification</td>
</tr>
<tr>
<td></td>
<td>Computer systems overview/prog. specs/database design</td>
</tr>
<tr>
<td></td>
<td>Procedures manual/forms</td>
</tr>
<tr>
<td></td>
<td>Software &amp; documentation</td>
</tr>
<tr>
<td></td>
<td>Trained users</td>
</tr>
<tr>
<td></td>
<td>Trainer computer dept. staff</td>
</tr>
<tr>
<td></td>
<td>Trained clerks</td>
</tr>
<tr>
<td></td>
<td>Database</td>
</tr>
</tbody>
</table>

Figure 5 — The Systems Development Cycle and Resulting Elements in the Operational System
### Information Inputs to the Development Cycle and Resulting Elements in the Operational System

<table>
<thead>
<tr>
<th>INFORMATION INPUTS</th>
<th>SYSTEMS DEVELOPMENT CYCLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Business Environment</td>
<td>Task Being Considered</td>
</tr>
<tr>
<td>➔ ➔ ➔ ➔</td>
<td>➔ ➔ ➔</td>
</tr>
<tr>
<td>➔</td>
<td>➔</td>
</tr>
<tr>
<td>➔</td>
<td>➔</td>
</tr>
<tr>
<td>➔</td>
<td>➔</td>
</tr>
<tr>
<td>➔</td>
<td>➔</td>
</tr>
<tr>
<td>➔</td>
<td>➔</td>
</tr>
<tr>
<td>➔</td>
<td>➔</td>
</tr>
<tr>
<td>➔</td>
<td>➔</td>
</tr>
<tr>
<td>➔</td>
<td>➔</td>
</tr>
<tr>
<td>➔</td>
<td>➔</td>
</tr>
<tr>
<td>➔</td>
<td>➔</td>
</tr>
<tr>
<td>➔</td>
<td>➔</td>
</tr>
<tr>
<td>➔</td>
<td>➔</td>
</tr>
<tr>
<td>➔</td>
<td>➔</td>
</tr>
<tr>
<td>➔</td>
<td>➔</td>
</tr>
<tr>
<td>➔</td>
<td>➔</td>
</tr>
<tr>
<td>➔</td>
<td>➔</td>
</tr>
<tr>
<td>➔</td>
<td>➔</td>
</tr>
<tr>
<td>➔</td>
<td>➔</td>
</tr>
<tr>
<td></td>
<td>WORKING PAPERS</td>
</tr>
<tr>
<td>----------------</td>
<td>----------------</td>
</tr>
<tr>
<td><strong>Production (P) or Control (C)</strong></td>
<td></td>
</tr>
<tr>
<td>Post-Implementation Report</td>
<td>←</td>
</tr>
<tr>
<td>Report on Systems Testing</td>
<td>↑</td>
</tr>
<tr>
<td>Cost/Value Justification</td>
<td>←</td>
</tr>
<tr>
<td>Feasibility Report</td>
<td>←</td>
</tr>
<tr>
<td>Specification of User Needs</td>
<td>←</td>
</tr>
<tr>
<td>Systems Specification (Overview)</td>
<td>←</td>
</tr>
<tr>
<td>Cost/Value Justification</td>
<td>←</td>
</tr>
<tr>
<td>Security Specification</td>
<td>←</td>
</tr>
<tr>
<td>Computer Systems Overview/Prog. Spec./Database Design</td>
<td>←</td>
</tr>
<tr>
<td>Procedures/Manual/Programs</td>
<td>←</td>
</tr>
<tr>
<td>Software and Documentation</td>
<td>←</td>
</tr>
<tr>
<td>Trained Users</td>
<td>←</td>
</tr>
<tr>
<td>Trained Computer Department Staff</td>
<td>←</td>
</tr>
<tr>
<td>Trained Clerks</td>
<td>←</td>
</tr>
<tr>
<td>Database</td>
<td>←</td>
</tr>
</tbody>
</table>
which an element draws information from a second element and an information input is required for the latter, and therefore also for the former. By considering this chart, one is able to observe the effects of a change in information inputs upon systems elements.

The systems development cycle can be seen as a structured procedure for producing elements in a system given certain information inputs. A similar, if not identical, chart will need to exist to process changes to a system. Figure 7 portrays the situations in which change in an information input will demand change in the cross-referenced elements. Not all information input changes will result in corresponding changes in cross-referenced elements. For example, changes in an installation policy might not affect users, hence user retraining may not be necessary. However, every element will need to be considered for every change in a cross-referenced information input to check if it is relevant.

In summary, a system consists of elements which were produced as a consequence of information gathered at the development stage. It must follow that changes to this information may demand alterations to the system.
Survey of the System Maintenance Literature

This section will discuss the maintenance literature related to each element in an operational system. Often certain elements are combined in the literature for discussion purposes; this practice will be continued here. Secondly, this section will consider the literature related to the organizational aspects of maintenance.

Specification of user needs

Many authors have addressed themselves to the problem of defining user needs. Many conceptual and practical techniques have evolved to address this problem. Davis [8] reviews a number of methods of definition and classifies them as belonging to one of the following strategies: asking, deriving from an existing system, synthesizing from characteristics of the utilization system, or discovering from experimentation with an evolving system. The first three strategies might well be termed engineering approaches as they utilize the basic philosophy of engineers, namely to compile a complete and detailed specification of the task prior to commencing development. They are based upon the assumption that a set of requirements can be fully determined prior to detailed development.

The fourth method, known generally as an evolutionary approach, recognizes the managerial problems involved in report specification and involves iterations toward a final system. Interestingly, an ideal system that satisfies managers' needs is usually implied after which, presumably, systems maintenance personnel requirements.

"Once the output system is shaped to accurately fit user requirements, an input system is designed and developed," Berrisford and Wetherbe [2]. McCosh et al. [22] in a similar way, note that "once the prototype has served its experimental purpose, it will be replaced by a robust, viable system." Evolution is usually not seen as a continuing process, but as a method of defining user needs ultimately resulting in a defined set of requirements which will form the basis for a processing system.

Research recognizing the continuing need to update user needs is not common. Edwards [11] reports empirical investigations designed to determine organizational variables that will encourage the continuing refinement of user needs through time. He envisages the maintenance of user needs to consist of amending the information provided to satisfy changing needs of user managers, ceasing production of non-used information, and encouraging usage of information perceived as underutilized by the users' superiors or the information's producers. (The term producers will be used here to denote staff of the organizational unit responsible for developing the information system and for producing the information.)

Edwards notes that the organizations in which the maintenance of user needs was operationalized all had the following characteristics:

- resources clearly devoted to maintenance of user needs,
- open user/producer communication channels, and
- allocation of responsibility to motivated personnel who perceived the maintenance of user needs as an important activity.

The crudeness of this research can be seen in the implicit assumption that users know what information they need. Such an assumption is also inherent in the evolutionary approach and usually implicit in distributed systems development. This conflicts with the human information processing literature summarized by Davis [8] which suggests that managers either cannot identify or are unable to express their needs.

Land [18], in synthesizing the work of others, suggests approaches to dealing with changing user requirements. His systems include considerable prethought of possible changes at the design stage, prototyping, and distributed systems development.

Ramsgard [28] provides another view by suggesting that an inventory containing all the essential facts relating to all currently produced reports will reveal redundant and irrelevant reports and highlight unreasonable distribution lists. Further, it will expose executives who receive too great a volume of reports.

The Comptroller General in the Standards for Audit of Government Organizations, Activities, and Functions [6] states that one of the general objectives of audit work is to ensure that financial
reports contain accurate, reliable, and useful financial data (emphasis added). This suggests the need to inquire if the organization is giving due consideration to identifying work (for example, producing reports) which serves little or no useful purpose. Internal auditors appear to view themselves as a checking function to ascertain if the organization is monitoring changing needs, however they suggest themselves as the secondary check, not the primary system.

Cost/value specification

The comparison of cost and value that is required prior to systems development will differ in content significantly from that which is undertaken to justify systems amendments. This, in turn, will significantly differ from the ongoing comparison to ensure that the value yielded is continuing to exceed cost.

Without some form of communication, managers would not be aware of the cost. Hence, Bernard et al. [1] suggests that a system of charging out cost to users would provide communication. This might encourage users to compare cost with the value yielded and to discontinue those reports that did not justify the cost of production. Drury [10] suggests that the existence of a charging system will encourage users to be economical in using the system. Users might well have difficulty in valuing a report, but may find it somewhat less demanding to decide if the value exceeds a stated cost.

The selection of appropriate costing methods is discussed in McKell [23]. The reoccurring costs of producing a report will form only a small part of the total cost as projected in the initial systems appraisal. This occurs mostly because of the hidden nature of development costs and the costs of database updates that would not be significantly reduced if a single report were eliminated. The relevant ongoing costs are a subset of those identified in the initial appraisal and, given that these are initially identified, can become the chargeout costs. Effectively, the cost/value justification is maintained to ensure value exceeds cost through the life of the system.

No literature was located that discussed organizing chargeout systems as part of a maintenance function.

Security specification

Many authors address themselves to the issue of designing systems that assure availability,
preserve confidentiality, and provide information integrity. There is little discussion in the information systems literature relating to the methods by which these controls might be maintained in changing organizational circumstances. The very detailed work by Martin [26] covers the issues in the area of designing secure systems. In 626 pages, Martin addresses 10 pages to this area and even these relate only partly to the issue of maintenance.

The issue of ongoing operations being reviewed on a functional basis is particularly relevant here. Issues such as protected areas, fire hazard, sabotage, electromagnetic radiation, and database recreation systems are a small sample of the issues to be considered in a functional perspective.

Auditors are becoming increasingly involved in the data processing function. Even so, many audit managers believe that they have only scratched the surface of computer auditing and do not have the personnel or audit tools they need for effective operation. Macchiaverna [24], in a Conference Board Report, states that EDP auditing was the most frequently mentioned problem area, receiving a mention by 16% of the 284 audit managers surveyed.

The role of the internal auditor in the maintenance of system controls presents an important issue. Figure 8 represents the systems related tasks shown through time. Main et al., [25] representing The Institute of Internal Auditors, see computer auditing as the verification of controls in three areas of the organization: applications, systems development, and the information processing facility. They suggest that the internal auditor perform function B in Figure 8 (verifying that adequate security controls exist), but that someone else should perform function A (monitoring the system and amending the controls as necessary). A Conference Board Survey [24] shows that 88% of internal auditors consider their role to include the review of data processing controls, but 70% also become involved in the development process to simplify subsequent auditing. They emphasize the role of review as against design to avoid compromising the checking function.

No literature was located that dealt with function A. Possibly producers who are pressured and overworked have been content to leave the task of identification of problem areas to internal auditors. With the increasing use of sampling on the part of auditors, this could be dangerous.
**Manual procedures specification**

The literature relating to the design of non-computer aspects of computer systems (for instance, the design of forms) grew out of the organization and management literature with slight adaption for the differences involved in computer processing. The literature has not developed independently. That which does exist is primarily concerned with systems creation virtually to the exclusion of maintenance, with repeated mention of the need for regular reviews.

**Database**

It is clearly important that data items and transactions be added to, amended, and deleted from the database. It is of such importance and so clearly vital to operations that systems are designed (at the development stage) to maintain the database. The ease of usage of such systems is a major attraction to DBMS. The problems encountered in adding data structures and changing the format of existing structures has been considered at length. However, such considerations seldom venture beyond the immediate systems element and hence integration of elements is slight. The major exception to this is the security aspects incorporated into database systems.

This is one of the few systems elements in which maintenance is designed into the system from the outset. As database management systems have developed, the implementation of changes both to the data and to the structure of the database have eased considerably.

**Training**

The training area has not changed significantly with the introduction of computer-based systems apart from the recent innovation of including training and assistance information within an application program to aid clerks and users alike. The content of training programs, more often than not, has focused on potential or existing systems analysts and programmers, although some do focus upon user managers. Such training for user managers is often not organizationally centered nor systems specific.

Seldom is the continuing availability of personnel trained in particular application systems considered in the literature. A survey of data processing managers undertaken by Lientz and Swanson [19] identified inadequate user training as one of the six major problem areas in maintaining application systems. They suggest ongoing training as an important element of the maintenance function. Ramsgard [28] discusses the use of an inventory containing samples of all reports as a training aid. He suggests that no new manager should be allowed to receive any computer reports until he has reviewed in detail the inventory of reports.

**Specifcation and documentation**

This area of maintenance is of great practical importance and has attracted academic and professional attention because it is virtually unavoidable. Producers do not need to monitor users' changing needs (users will inform the producer in no uncertain terms if the changes are important). Nor do producers need to evaluate systems security (again, problems will become very obvious or Internal Audit will inform the producer of any inadequacies). But producers do need to correct a system that fails. One must consider, however, that corrective software maintenance is usually regarded as a small part of the total software maintenance task (between 19 and 21% of the total software function [19, 20]). Other reasons for software maintenance — perfective and adaptive — will also be difficult to avoid. User pressure for systems improvements can become very intense. In other cases, a new model of computer will demand software changes before it will process existing applications.

The literature can be classified into four groups or issues.

**Cost of Maintenance** — A search of the literature failed to locate studies that isolate the cost of systems maintenance as defined here, but a number of investigations highlight a subset of this — the cost of software maintenance [5, 13, 16, 17, 19, 29, 32, 33].

**Tracing, Recording and Costing Software Amendments** — These range from very simple [34] to the more complex computer based systems [12] and are directed toward software maintenance cost analysis, [1, 32, 33], general
related statistics [27, 29, 30], and security aspects [31] of implementing changes to software.

**Motivation of Software Maintenance Staff —**
The benefits accruing from integration or separation of maintenance from development forms the focus of attention.

**Improved Programming Techniques to Ease the Maintenance Problems —**
The United States General Accounting Office [33] identified the improved use of tools and techniques as the second most effective way to reduce maintenance costs. Ewers and Vessey [12] discuss 15 programmer productivity tools which they classify under the headings of: tools used in the programming environment, tools used in the program creation function, and program testing tools. They especially emphasize the maintenance benefits from such productivity tools. Donahue [9] uses over 100 pages to extensively describe software maintenance tools and techniques in great detail. Discussions of software tools tend to be technical, and utilization often involves the purchase of particular software aids marketed by computer manufacturers and other large software producers.

**Resource requirements**

If producers are to play any significant role in maintenance, resources will be required. Lientz et al., [20] report that most senior managers regard software maintenance as more important than development and hence, if the survey was representative, resources may well be made available for software maintenance. Glass and Noiseaux [15] profile a software maintainer which demands the qualities of a superman: flexibility, broad background, patience, self motivation, responsibility, humility, innovation, and an historian. Authors agree that, in reality, the most inexperienced, the most inept, or the person most easily pressed into service is often utilized.

**Organizational design**

Tipshus and Sanderson [31] suggest a novel form of organization for software maintenance. They suggest creating a project team for development consisting of transients (who will move on to other systems when the demand for resources is reduced), and fixtures (who will stay with the system on a more or less permanent basis). Knowing from the outset that one is to be involved in the maintenance of a system may well encourage one to develop adequate documentation. Freedman and Wiener [14] suggest adding maintenance staff to the post-implementation review committee to achieve this same objective. Cooper [7] suggests a variant of this whereby a permanently assigned program maintainer joins the development group to ease the transitional problems. Lientz and Swanson [19] report that separation of maintenance and development leads to increased efficiency, although this was by no means the usual method of organization noted. Other cases of alternative organizational methods can be found [31].

**Communication channels**


**Discussion**

This article has attempted to show that systems maintenance should involve all elements in an operational system. A review of the literature reveals a dearth of literature for the majority of the elements of systems maintenance and a nearly total fragmentation into non-interacting specializations. This section offers positive direction in the maintenance dilemma by integrating proposals from many sources.

**Why is the present situation so unsatisfactory?**

The question of why this situation is presently unsatisfactory must be addressed before becoming involved in a consideration of action. Organizations spend thousands, if not millions of dollars developing computer-based information systems
with the intention that any individual system will last for a number of years. Clearly, information support for the managerial tasks must have been important when the system was first developed and, unless a major change in organizational tasks and roles has occurred, information support will still be of importance. As time passes and organizational changes occur, the system may yield progressively less value. The users, having become familiar with the original system, will possibly be requiring systems enhancements to increase the value yielded. Hence, the gap between the value potentially and actually yielded becomes wider. Awaiting the obsolescence of a system before developing a replacement is unsatisfactory. For most of the life of that system, it will not be providing the required information, and extremely significant problems could occur by not keeping the system current. For example, breaches of security could occur resulting in the misplacement of millions of dollars. Liken this lack of maintenance to the purchase of any other organizational asset that depreciates, say a machine or a new building, and the need for systematic maintenance becomes patently obvious. A continually evolving system to meet changing organizational circumstances would appear preferable to a quickly degenerating asset. In addition to these major points, the aggravation involved in users having to utilize an unsatisfactory system can hardly be productive.

All of these comments addressing the need for maintenance hinge upon change occurring in the organization through time. Even though some systems in an otherwise changing organization might not be effected Lincoln [21] identifies a number of organizational and environmental pressures that would suggest that static organizations should be most uncommon.

Selecting a philosophy

Figure 7, which cross references inputs to and outputs from the systems development process, is actually cross referencing two possible focuses of the maintenance function. First, a maintainer could adopt a reactive philosophy which involves awaiting demands for maintenance and adjusting the system's elements to satisfy the specified change. Such a method is effectively reacting to changes that become apparent. Programs would be amended if they fail, improved if changes are pressed by users, security procedures would be tightened after a failure becomes apparent, user training would be available upon request. Such a system would be proportionately inexpensive to operate, but the value yielded by the system may also be similarly small.

An alternative philosophy, here termed proactive, would involve monitoring changes that occur in the information inputs to the design process and, by using a chart of the form of Figure 7, tracing the effects of each input change upon the elements in the system. Such a system would, for instance, demand the incumbent of a position to reappraise each report received for its utility when a particular organizational position is changed. Further, even if a manager filling a position is not replaced, he will need to be monitored to ensure that his needs, upon which the reports were designed, have not changed.

A combination of these two philosophies is clearly possible, whereby certain systems elements are considered on a reactive basis and others are monitored on a proactive basis. For instance, users might be monitored on a proactive basis whereas software might be organized on a reactive basis.

Selecting a maintenance philosophy will depend primarily upon the projected degree of change in the environment of the system and the costs of alternative maintenance systems. Such variables are difficult to quantify. The projected degree of change and its projected effects can best be found by undertaking appraisal of each information input to the development process. Certain applications addressing particular tasks (for example, word processing) appear to require change less frequently than systems that are report specific (for example, order analysis reporting). The consequences of an ill-suited information system can vary widely. The cost of alternative systems of maintenance, although not simple, is somewhat easier to appraise.

When to consider maintenance

Maintenance should be considered at the system's design and creation stages: it should not be an afterthought. Either a separate task in the development cycle titled 'design maintenance procedures' is originated or a part of every
systems element should be devoted to maintenance. This is impossible for existing systems, but no further systems proposals should be drafted or accepted without including a clear consideration of how each element is to be maintained. Systems design of the maintenance function includes not only a discussion of what is required to be performed but also who is to do it and how it is to be organized.

Allocating responsibility

The design of maintenance procedures should be the responsibility of the system designer. Responsibility for the maintenance function can be performed by users, by producers, some by either, and some by both. As a starting point, it would seem reasonable to allocate the responsibility for each element to the group that was responsible for producing it. This would imply that if system analysts were involved in defining user needs, they would be continually involved in noting changes that affect those needs. To turn responsibility for maintenance over to users, as is implied when authors suggest ‘signing off,’ requires that the users have learned a great deal from their participation in the development process. Further, it implies that they have the required degree of overview of the whole system to realize the implications of change. If such a turnover of responsibility is envisaged, exhaustive user training will be needed. Lientz and Swanson [19] suggest that enhancements in terms of additional reports could be processed by users with the use of report generator languages whereas enhancements that demand changes to the database would be processed by producer staff. Whatever the division of responsibility, it is vital that someone has responsibility for the success of maintenance just as the leading systems analyst has in the development function.

Resource requirements

There is a need for senior management to recognize the need for, and importance of, systems maintenance in order to allocate adequate resources. Previously discussed research suggests that senior managers do recognize this need. The extension of these perceptions from software maintenance to systems maintenance does not necessarily follow, although the logic underlying the importance of systems maintenance might well encourage support.

Developing communication channels

Systems maintenance depends on open communication channels between producers and users. Communication can be achieved in a variety of formal and informal ways, but the method of physically locating an analyst with a group of users to service a single system appears to meet the objectives of a close contact with users. Yet, in extreme cases, this may isolate analysts from producers. Many other organizational possibilities exist to facilitate communication: for example, systems discussion groups, regular systems reviews, informal activities, and advanced training sessions.

The evolving role of the internal auditor

The internal auditor has been mentioned in relation to the maintenance of various elements. The obvious extension of this responsibility was suggested by Lientz and Swanson [19] in their recommendation for life cycle audits. They did not contrast the roles of maintenance as an ongoing task, and audit as a final checking function. Rather than being seen as a first line maintainer, the internal auditor should be seen as the final check on efficient and effective operation by investigating all systems elements. To safeguard compromising their position, auditors should not become associated with the systems maintenance function.

Attitude changes

Two related, but apparently minor points might be valuable in securing effective systems maintenance. First, the title maintenance hardly engulfs the wider organizational tasks envisaged in this article. Such a title suggests merely striving to ensure that an initially given value persists. Systems evolution, or some similar title, might move away from the narrow meaning of maintenance, toward a definition of striving to increase the value of the system to the organization. A sec-
Information Systems Maintenance

The effects of distributed data processing upon the maintenance functions

A final point worthy of consideration is the effects of distributed processing upon the systems maintenance function. There are many shades of distribution ranging from near centralization of all systems elements to full user autonomy in creating and operating systems. The wide variety of situations that are labelled as distributed processing would lead to an extensive discussion of the effects of distributed data processing upon the maintenance activity. Such a discussion would not be suitably presented in this paper, but global comments might be appropriate.

The control of elements by a single department head will likely lead to an advantageous trade-off between benefits and costs. The benefits of systems maintenance in terms of improved information will be known to the operating department head, in addition to the costs. A choice as to resource allocation can be made based upon full information and with less political interference. The monitoring of either systems elements, or information input, will be considerably simpler to organize given it is the responsibility of a single person. The changes are occurring in the department in which the maintainer is employed, and hence the communication problem should be reduced.

Reasons for the Neglect of Maintenance

One major question remains to be addressed: why, if systems maintenance is so important, is it so neglected in practice and academia?

Considering the former, four points appear relevant. Firstly, it is often noted that analysts and programmers prefer to be concerned with the development of new systems rather than the maintenance of existing systems. Glass and Noiseux [15] include a discussion of six major points why analysts and programmers are not attracted to maintenance:

- maintenance is intellectually very difficult
- maintenance is technically very difficult
- maintenance is unfair; necessary information is not available; original program writers get promoted even though they leave a mess behind them
- maintenance is no-win; people only come to maintenance with problems
- maintenance work does not result in glory, noticeable progress, or chances for "success"
- maintenance lives in the past, the quality of yesterday's coding is often very poor

These points combined with a world shortage of trained staff does not encourage senior managers to insist on maintenance; computer staff are all too mobile.

Second, it may not be in an analyst's interests to add all of the very significant cost of maintenance into the cost/value proposals for risk of fewer projects being undertaken. The underestimation of maintenance cost in proposals makes it somewhat difficult to request additional staff once systems are implemented. It is easier to severely limit the activity to that which is absolutely necessary. Any additional costs can be hidden either in the development of subsequent systems or through the non-analysis of costs; a practice that has been noted [33]. Comparing projected with actual maintenance costs would demand a sophisticated accounting system and even then many reasons could be found for the huge difference. If, by chance, such information was available, an analyst would be even more tempted to do less maintenance.

Third, when a department experiences very high service demands with a relatively fixed budget something has to suffer; the usual attitude in many walks of life is to select aspects which are least pressing from a short-term viewpoint and most difficult to measure. Systems maintenance, apart from software correction, appears to be an obvious choice.
Lastly, maintenance has not been accepted historically as a major task. The author can remember when, working as a junior programmer, a discussion centered upon the fate of the programming staff when all the organizations systems were fully programmed: viewing maintenance as a minor function is all too prevalent. To change such perceptions demands a major effort that may take a considerable period of time.

These foregoing arguments may suggest why maintenance is neglected in practice but does not explain why the area has received so little attention from academics. Apart from a very small number of noted people, research on the maintenance issue has been neglected. This applies, to a lesser extent, to all areas of MIS research. As in other areas of academia, there is a tendency to research where a body of knowledge already exists (see, for instance, the MIS research developing out of cognitive psychology [34]). To be among the few working in an area is unfashionable and often leads to papers such as this where the discussion can only suggest the most tentative of actions.

Acknowledgements

My thanks are offered to John Handley for many helpful comments offered on an earlier draft of this article.

References


About the Author

Professor Chris Edwards (MA PhD FCMA) is a Professor of Management Information Systems at the Cranfield School of Management. He taught at the University of Strathclyde between 1973 and 1981. From then until 1983 he was researching and teaching at the Carnegie-Mellon Graduate School of Industrial Administration. Professor Edwards joined Cranfield School of Management in 1983. He is particularly interested in the organizational problems encountered by businesses using microcomputer-based information systems. He is also involved in research related to the effectiveness of different types of information presentation, in particular the effective use of graphics.