Payoff

A formal procedure for managing maintenance activities opens a window of opportunity for the IS manager to consider systems as candidates for reverse engineering, reengineering, and restructuring. This article outlines a software maintenance methodology. It discusses development techniques (e.g., the use of CASE tools and standard data definitions) that help to ensure software maintainability and the use of a change management system that enforces more productive maintenance procedures.

Problems Addressed

Software maintenance is sometimes viewed as a necessary evil—a laborious, uninspiring, and costly task. Nonetheless, maintenance must be performed over the life of a system, which may run for years. Maintenance can be beneficial and well managed, or it can be burdensome and expensive. This article suggests methods for making software maintainability a paramount goal of systems development.

Types of Software Maintenance

Software (and hardware) maintenance can be categorized into four types:

- Corrective maintenance.
- Adaptive maintenance.
- Perfective maintenance.
- Preventive maintenance.

Corrective Maintenance

Corrective maintenance is the most burdensome part of software maintenance because it corrects design, coding, and implementation errors that should never have occurred. Commonly, corrective maintenance is the result of an urgent or emergency condition that needs immediate attention. The ability to diagnose rapidly and remedy the error or malfunction is of considerable value to the organization.

The need for a significant level of corrective maintenance usually implies that the systems development life cycle (SDLC), various systems development modeling tools and technologies, and testing procedures were not used properly or at all while the software was being built.
Adaptive Maintenance

Adaptive maintenance is performed to satisfy changes in the processing or data environment and to meet new user requirements. The environment in which the software operates is dynamic; therefore, the software must be responsive to changing user requirements. For example, a new tax law may require a change in the calculation of net pay, or the adoption of a new accounting depreciation method must be installed, or a report's content and format need to be updated. Generally, adaptive maintenance is inevitable. Too much of it indicates, however, that user requirements were not defined adequately during systems development.

Perfective Maintenance

Perfective maintenance enhances performance or maintainability and meets user requirements that had gone unrecognized. When making substantial changes to any software module, the maintenance staff also exploits the opportunity to upgrade the code, remove outdated branches, correct sloppiness, and improve documentation. For example, this maintenance activity may involve reengineering or restructuring software, rewriting documentation, altering report formats and content, and designing more efficient processing logic. The need for perfective maintenance, as for adaptive maintenance, may also indicate that analysts did not discover all user needs or that programmers were unable to fulfill all requirements before the software was released.

Preventive Maintenance.

Preventive maintenance—also called proactive maintenance—consists of a periodic inspection and review of the system to uncover and anticipate problems. As maintenance personnel work with a system, they often find defects that signal potential problems. These defects may not require immediate attention; if left uncorrected in the minor stage, however, they could significantly affect either the functioning of the system or the ability to maintain it in the near future.

Improving Software Maintainability

Maintainability refers to the capacity of maintenance personnel to perform corrective, adaptive, perfective, or preventive maintenance without wrestling with unnecessary obstructions, such as poor design and the lack of documentation. Clearly, these people will have difficulty performing maintenance if the software cannot be understood.

Maintainability should be a paramount goal of software development. The following section discusses ways to reduce obstructions and build applications software that is highly maintainable.

Designing for Software Maintenance

Software maintainability is increased if the system is designed to make changes easier. A maintainable system is more flexible because it allows new features to be incorporated easily, enables programmers to locate and resolve problems, and prevents a change from producing unforeseen side effects. Maintainability encompasses a variety of procedures, some of which are explained in the following paragraphs.
SDLC and SWDLC.

The professional application of the systems life cycle and its subcomponent, the software development life cycle (SWDLC), establishes an engineered, structured approach to the development of the total system and its supporting customized applications software. Various modeling tools (e.g., data flow diagrams, entity-relationship diagrams, and structure charts) are used to facilitate the SDLC and SWDLC. The application of these modeling tools can be automated using Computer-Aided Software Engineering technologies.

Standard Data Definitions.

The trend toward data base management systems underpins the need for standard data definitions and data normalization. Redundant and inconsistent data definitions that exist throughout an organization's procedure manuals, source program documentation, and data files add further to the problem of maintenance. A glossary or data dictionary of terms for data elements and other items in the system should be provided. For example, all data elements should have a standard name, description, size, source, location, security, and maintenance responsibility designation. It is also important to use the name precisely in each application. CUST-NAME is not the same as CUSTOMER-NAME, for example.

Standard Programming Languages.

The use of a standard programming language, such as C or Common Business Oriented Language, makes the maintenance task easier. If COBOL- or C-based software contains complete and clear internal documentation, even a novice maintenance programmer can understand what the software does. Moreover, C and COBOL are universal languages generally known to many people. Therefore, maintenance programmers turnover has less impact on the company's ability to maintain old C and COBOL programs.

Modular Design.

With home appliances, a repair person can determine which part is causing trouble and quickly replace it. Similarly, maintenance programmers can change modules of a program much more easily than they can deal with the total, monolithic program. Many object-oriented and nonprocedural language make true modularization difficult. The flexibility of these languages can be constrained, however, by developing standards to group logically related blocks of code to facilitate debugging and maintenance.

Reusable Code.

Modularity fosters reusability. One common module of reusable code can be accessed by all the applications requiring it. Should it need to be changed, modifying this one common module is all that is needed. Computer-Aided Software Engineering products provide a central repository in which previously defined data and program modules can be stored and accessed for reuse—a key productivity aid for systems developers. In fact, storing data and program modules in a central repository or library for ready access is a key feature of systems development and software maintenance.
Documentation Standards.

Complete documentation is needed (for the system, the users, the programs, management, and operations) so that all the information required to run and maintain an application is available. Documentation must be up-to-date with the code in use. This supportive material should include reference to the changes that have been made to the code. Ideally, changes to code should automatically generate supportive documentation and data that can be assembled in management reports. Effective documentation can include Computer-Aided Software Engineering tool products, flowcharts, operator's manuals, and program source code. Although maintenance activities are initiated by many sources, much of the documentation effort focuses on the maintenance programmers.

Central Control.

All programs, documentation, and test data should be installed in a central file under the control of a directory and program librarian package. (A Change Management System that accomplishes this job and more is discussed later in the article.)

Test Case Libraries.

Test cases that were created during the testing phase of the SWDLC should be retained in a library for maintenance work. Some test cases can be turned over to maintenance programmer; others will be used by the quality assurance group.

Standard Fourth-Generation Design Aids.

A software package built years ago may no longer be the best means of processing some applications. A more easily maintained solution may be to scrap the old software and to developing a new system using fourth-generation languages (4GLs), data base query language (e.g., Structured Query Language and Query-By-Example), or function-specific software (e.g., a spreadsheet or a report generator) that is available off the shelf.

CASE and 4GL tools serve as aids to automating the project development life cycle. Indeed, systems analysis, design, construction, and testing are supported by assorted available Computer-Aided Software Engineering and 4GL toolsets.

Organizing for Software Maintenance

There is no clear-cut, best way to organize for systems maintenance; however, three methods are worth exploring. The choices are to:

· Separate systems development and software maintenance.

· Combine systems development and software maintenance.

· Position systems professionals who have responsibility for both systems development and software maintenance within the organization's functional areas.
Separating Development and Maintenance

Traditionally, IS programmers and analysts have been organized into two distinct groups: development and maintenance. This separation of duties provides a natural way for one group to force the other group to perform its work properly. For example, a maintenance programmer would not accept a new program for operations unless it had been thoroughly tested during development.

Maintenance that is performed separately from development has its advantages; it requires better-prepared documentation and formalizes change procedures as well as the system's conversion to an operational system. In addition, senior maintenance programmers may be promoted to development project leaders, because they have acquired in-depth knowledge of documentation requirements, standards, and operations. Maintenance positions are also an effective training ground for junior programmers.

A Combined Approach

The combined approach brings together both development and maintenance personnel into one major group within the IS department. This form of organization creates a closed loop in which systems developers are forced to deal with all maintenance problems as well—a situation that encourages the use of development techniques that ensure the software's maintainability.

If both development and maintenance are performed in the same group, then users have one point of contact with IS personnel who can effect change. User departments often do not know whether a request for work will be classified as development or maintenance, because large revisions or system improvements are often treated as development. Furthermore, the analysts, designers, and coders who originally developed the systems can assess the full impact of changes. Some software is so critical and complex that maintenance must be handled by only the most capable people, and in many cases, the most capable people are those who developed the software in the first place.

The Functional Approach

The functional approach is a variation of the combined approach. The difference is that the functional approach removes systems professionals from the IS department and assigns them to business departments, where they are responsible for both development and maintenance. Specialization is by the organization's functional areas, and is no longer a choice between development and maintenance. The functional approach to organization puts systems professionals much closer to users, in terms of both physical proximity and knowledge of users' jobs and requirements.

By virtue of employing the functional approach, systems professionals are actually members of the user departments and, consequently, have a vested interest in the success and feasibility of systems applications. Moreover, systems professionals become more business-literate and users become more systems-literate. The payoff is understanding, communication, and mutual interest. The downside of the functional approach includes a possible loss of IS control over development standards and procedures and blurring of priorities.
Software Maintenance Methodology

If the development of software requires a structured methodology, then it seems logical that after the software has been converted to operations, its maintenance should also follow a structured methodology. A number of authorities recommend that software maintenance be performed according to a software maintenance life cycle (SMLC). Essentially, the Software Maintenance Life Cycle includes the following phases.

The Maintenance Request

The user submits a maintenance request that is used to prepare a maintenance Work Order (see Exhibit 1). The work order includes information that management is interested in: the work requested, work performed, estimated time versus actual time, maintenance code, and maintenance cost. The maintenance code is structured as illustrated in Exhibit 2. Accordingly, maintenance code 116, for example, describes corrective maintenance with emergency priority and a modify activity.

Maintenance Work Order Form

Deciphering the Maintenance Code

Transforming the Request to a Change

The maintenance staff has a description of both the existing software and the desired software. Transforming the request to a change involves identifying and eliminating the differences between the two.

Specifying the Change

The change may involve all or parts of the existing code, data, and procedures. The new code, data, and procedures are sometimes termed a patch.

Developing the Change

The change of patch is designed and coded.

Testing the Change

Test procedures are applied to the software after it has been changed. Testing helps to validate and verify that the right change has been made and made correctly. Code walkthroughs, similar to testing in the software development life cycle, focus on the modified code itself (i.e., does the changed code work?).

Regression testing (sometimes referred to as revalidation testing) is done to confirm that modules of the software that were to have been left unchanged still perform as they were intended to perform before the required change. Regression testing focuses on the functional integrity of the total software package after the change.

Training Users and Running an Acceptance Test

If the change is relatively simple, this step may be skipped. If the change introduces new ways in which users perform their tasks, the users must receive training. After users have been trained, acceptance tests should be conducted using alpha testing—that is, usability or user testing in which users perform their own tests while the maintainer is present.
User involvement in the testing phase may elicit calls for still more changes. These changes should be processed as new work order.

**Converting and Releasing to Operations**
When the newly changed software has successfully passed all tests, it is ready to become operational. Simultaneously, the old version of the software should be withdrawn according to a conversion plan.

**Updating the Documentation**
All the documentation pertaining to the maintenance activities should be updated to reflect the newly changed software. Equally important, the out-of-date documentation should be withdrawn from service to avoid confusion.

**Conducting a Postmaintenance Review**
The software should operate for a few days or weeks after maintenance has been performed. At that time, the maintenance staff should conduct a postmaintenance review to determine whether the change continues to meet user expectations.

**A Source of Management Information**
Not only does the software maintenance life cycle provide a systematic method of performing software maintenance work, it produces a wealth of information that helps management assess the maintenance process to determine where the trouble spots are and to optimize both development and maintenance efforts. Information on all changes that have been made to software may help management determine whether a program should be reengineered, for example. In addition, such information helps to identify problems in the organization’s software development life cycle, its programmer training methods, and in the standards the organization uses.

The key to generating such information is the maintenance work order form. Specifically, the form is used to:

- Compute a variety of maintenance cost analyses.
- Measure the number of failures per program.
- Calculate the total number of hours spent on each maintenance type.
- Compute the proportion of emergency, urgent, and routine maintenance.
- Derive the average number of changes made per program, per language, and per maintenance type.
- Develop a profile of those making most of the maintenance requests and the common problems encountered.
- Determine average number of hours spent per line of executable code to change, add, or delete such code.
- Calculate the average turnaround time per maintenance work order. This calculation entails the Mean Time To Repair.
Build a mean time between failure (MTBF) profile on all applications. From this profile it is possible to determine those applications that are consuming the largest portion of the maintenance budget and why.

Tools for Software Maintenance

Tools that support software maintenance fall into three categories: reverse engineering, reengineering, and restructuring.

Reverse Engineering

Reverse engineering must be performed in situations in which the appropriate development techniques were not applied. Reverse engineering is a process of examining and learning more about the existing software by recreating its design. The entire system is read, including source code, screens, reports, data definitions, and Job Control Language. The results are abstractions of design specifications in the form of models such as data flow diagrams and structure charts. From these resources, systems professionals can obtain a clear view of the old system so they can analyze and evaluate its design quality and capabilities. After the system has been fully evaluated, a decision may be made to scrap it.

Unfortunately, much of the software that was developed 10 to 25 years ago was poorly designed, structured, and documented. An important and logical question is whether IS should continue to maintain this old software. Sometimes scrapping the existing software and developing new software from scratch is more effective and efficient. Reverse engineering provides enough design-level understanding to reengineer or restructure the software for easier maintenance or to give management sufficient information to support a replacement decision.

Reengineering Tools

Some software may be worth reengineering. Reengineering generally includes some form of reverse engineering to gain a better understanding of the existing system, followed by redevelopment engineering to redesign and change the quality and capability of the software. Unnormalized data is normalized, data names are standardized, and ambiguities, redundancies, anomalies, and unused code are eliminated. Inefficient code is recoded to be made efficient. Unstructured code is structured. Documentation is prepared or improved. In essence, the quality and capability of the existing software are improved. The software is then reimplemented in a new form.

Reengineering is ideal for supporting major corrective, adaptive, perfective, and preventive maintenance projects. A substantial number of Work Order pertaining to a given system is indicative of a need to reengineer the system.

Restructuring

If the software's capability essentially meets the needs of the user, then the software is functionally sound. However, old software that is functionally sound is often structurally unsound, which makes the software difficult to understand and difficult to maintain.

The restructuring process converts unstructured spaghetti code into fully structured documented code. The restructuring (or more precisely, the structuring) of existing unstructured code is one of the simplest methods available to decrease the cost of maintenance.
Many standalone products or elements of Computer-Aided Software Engineering systems are available that enable existing Common Business Oriented Language programs to be structured automatically. These products input spaghetti code at one end and structured, documented code is produced at the other end. The resulting structured software preserves the capability of the original unstructured software and makes it easier and less costly to maintain.

**Employing a Change Management System**

Change management has become more important as the amount of software maintenance has increased. A comprehensive Change Management System assists in controlling the maintenance resources being deployed, in developing new software, and in maintaining existing software. A change management system can be purchased for many hardware platforms for as little as $500 for a microcomputer and for more than $100,000 for a mainframe. A change management system is used to:

- Restrict access to production source and object code.
- Reduce errors and design defects that may be introduced into the production system.
- Prevent the existence of more than one version of source- and object-code programs in the Production Master File.
- Increase security and overall control of software development and maintenance activities.

A change management system is illustrated in Exhibit 3. Its principal components are the Librarian Function Facility, programmer workstations, the Test Master File, the Quality Assurance Master File, the production master file, the Backup Master File, and management reports and audit trails. A maintenance Work Order initiates change management activities.

**A Change Management System**

**Librarian Function Facility**

The Library Function Facility is the heart of the change management system. It works in a manner similar to any librarian function, collecting and controlling vital documents and software.

The librarian function facility is itself a software package that centralizes, tracks, controls, and automates changes to software against an approved maintenance work order. It also controls the implementation of newly developed or acquired software. Exhibit 4 depicts the program promotion and release hierarchy. If a program that is already in production has to be changed, it is logged out to the test master file. If a new program is to be placed into production, it must first enter the test master file. No direct introduction of new software or changes to old software are allowed in the production master file.
Software Promotion and Release Hierarchy

The librarian function facility controls linkage between source and object code and automatically loads modules online for execution, thereby ensuring synchronization of both the source and object code. Comprehensive management reports and audit trails are available on screen or as hard copy for history, status, tracking, and performance information. All master file are backed up to safeguard the system from disasters. Programmers have online access to the change management system to augment change productivity. Programmers’ access privileges are controlled by passwords or biometric control devices.

Programmer Workstations

The change management system acts as a single point of control, and approved work order initiate log-outs and log-ins that allow programmers to do their job at their workstations. Log-outs are done against an approved and assigned work order.

When working against a specific work order, programmers are permitted to log out as many modules as necessary to complete the assigned unit of work. Program modules outstanding to other work order are noted during the log-out process.

Log-ins are performed to promote and release a changed and tested program to the quality assurance master file and then to the Production Master File. Careful testing and walkthrough procedures are conducted before any program or program modules are promoted and released to the production master file.

Middle-of-the-night production problems generally require a maintenance programmer to be onsite or, if at home, to return to the office to correct the problem. On-call programmers with workstations installed at their homes and connected to the change management system can increase the efficiency of maintenance work. Although this setup does not guarantee that programmers may not receive a late-night phone call, it may reduce the need for a trip to the office.

Test Master File

To change a production program, a work order must be opened and the program demoted from the Production Master File to the test master file. This operation changes the program from production status to test status. As long as the test status is maintained, the program can be changed.

A new program entering the test master file is subjected to testing procedures. If it passes these tests, it is promoted and released to the quality assurance master file. A production program that is demoted to the test master file has already been subjected to these tests, but after it is changed, it is subjected to regression testing.

Quality Assurance Master File

Some change management system employ an independent quality assurance group. This group serves as an additional control, because it reviews and tests the program independently of the maintenance programmers before the program is promoted and released to production. In addition to performing an array of tests, a chief function of the QA group is to conduct source code walkthroughs. In a source code walkthrough, the QA group reviews the actual program code to see whether it matches change requests, design
specifications, and standards. Then the group simulates how such code will be processed by the computer to discover coding errors or malfunctions.

**Production Master File**

After a program enters the production master file, it is locked into production status and cannot be changed. With proper authorization, a program can be copied and logged into the test master file with a new name, and the copied version can be changed. This protective feature helps to ensure that production programs will not be changed inadvertently.

**Backup Master File**

In the event that any master file is destroyed, the change management system permits IS personnel to recover from a backup master any files or specific modules that may have been lost. Similarly, if a program placed into production fails to operate as expected, recovery is enabled by the retrieval of a previous version. Usually, a copy of the backup master file is maintained locally and another copy is stored off site in a secure location.

**Management Reports and Audit Trails**

Reporting features help managers to develop an optimized change management system and auditors to attest to the integrity of the change management system. Management needs information about the maintenance process so that it can evaluate and optimize maintenance activities. Audit trails help to ensure the integrity of the system.

**Recommended Course of Action**

Software maintenance includes all changes made to a software product after it has been turned over to operations. Like any product, software must be changed as necessary to continue to satisfy changing user requirements. IS managers can take certain actions to make this task less expensive and time-consuming.

**Built-in Quality**

Corrective maintenance is the most expensive task in information systems development, and the reason for this resource-consuming task is that the software product was not designed and developed correctly in the first place. Correcting errors and design flaws during operations is the worst kind of software maintenance. The IS manager should therefore ensure that systems developers practice Total Quality Management and follow a well-developed SDLC. A major part of the SDLC implementation phase requires stringent testing procedures and design walkthroughs. Before becoming operational, all software products should be subjected to a comprehensive testing process and pass stringent quality standards.

IS managers can learn a lot by taking notice of what is happening in world-class manufacturing. Traditional manufacturers believed that quality increased costs. Today's world-class manufacturers have learned that quality actually decreases costs. The underlying concept is to design quality in, rather than try to build (or maintain) it in. World-class manufacturers recognize that as a product moves from design to production to consumers, the more costly it is to correct errors and defects. IS managers can develop world-class information systems (and easy-to-maintain software) by emulating their counterparts in the manufacturing sector by designing quality into their products.
Developing Maintainable Software
Generally, poorly designed software is not only expensive to maintain, but some of it is difficult or even impossible to maintain. Indeed, the high costs of software maintenance are in large part due to the need to maintain software that was developed without the use of structured methods and sound documentation procedures. Such software lacks proper modular structure and contains spaghetti code, so that it is difficult to change one part of the program without affecting other parts. As a consequence, it is hard to develop testing procedures to validate the modified program. Without comprehensive, current, and correct documentation, the maintainer may not be able to understand enough about the software to perform any maintenance. Computer-Aided Software Engineering products can recast a program from spaghetti and undocumented code into a structured and documented format. CASE products can reverse engineer, reengineer, or restructure old software so that it becomes much easier to maintain. IS managers should encourage the use of CASE tools if unstructured and undocumented software exists in their systems.

Organize for Software Maintenance
Maintenance costs are frequently increased by low morale on the part of the maintenance staff, with resultant low productivity. Usually, this unhappy and costly situation is a consequence of the lower status accorded by some organizations to maintenance work as compared with systems development. The job of software maintenance should not only be accorded high status, it should be organized properly.

Implement a Software Maintenance Life Cycle
Software development follows a software development life cycle. Because software maintenance should also follow an engineered methodology, the installation of a software maintenance life cycle(SMLC) is strongly recommended. Besides providing an engineered approach to performing maintenance, the Software Maintenance Life Cycle is instrumental in providing information that helps IS managers to analyze and optimize software maintenance.

Install a Change Management System
To support the foregoing actions in an efficient and effective manner, a Change Management System should be installed. A change management system centralizes, controls, tracks, and simplifies the development and maintenance of all computing resources, especially software. Additional IS management benefits delivered by a change management system include:

- Clearer separation of responsibilities.
- Avoidance of loss of changes.
- Improvement in both development and maintenance productivity.
- Synchronization of source and object code.
- Ability to back out of changes if necessary.
- Enforcement of quality assurance standards.
- Improvement in auditing, security, and control.
· Upgraded management reporting and review.
· Improved priority setting.

Bibliography


Author Biographies

John G. Burch

Fritz H. Grupe
Fritz H. Grupe is Associate Professor of computer information systems at the University of Nevada, Reno. His research interests are in expert systems, geographic information systems, and computer center management.
Maintenance Work Order

Work performed: Add vendor performance code to inventory record.

Name: Tom Barkley    Title: Purchasing Agent

Work performed: Added vendor performance code. Tested and documented change.

Name: Maria Gomez    Title: Maintenance Programmer

Changed Approved: Harry Feldman    Date: MM/DD/YYYY

Labor Cost (actual time x labor rate) $_________  ________

Material or Parts Cost

Total Maintenance Cost $_________  ________
Maintenance Code

Type of Maintenance
1. Corrective
2. Adaptive
3. Perfective
4. Preventive

Priority of Maintenance
1. Emergency
2. Urgent
3. Routine

Type of Activity Involved
1. Scrap
2. Reengineer
3. Restructure
4. Replace
5. Repair
6. Modify