AN ALTERNATIVE TO CHAOS OR MAINTAINABLE SYSTEMS

EARL J. DEMAREE

AIR FORCE DATA SYSTEMS DESIGN CENTER
MONTGOMERY, ALABAMA 36114

Keywords & Phrases

Corrective Maintenance
Preventive Maintenance
Maintenance Prevention
Availability
Capability
Maintainability
Reliability
Software Engineering
Systems Life Cycle

Abstract

One of the least talked about and most costly phases of the Systems Life Cycle is Maintenance. This paper gives an overview of maintenance and some alternative ideas on how we should address the subject. Topics like Reliability, Availability, Capability and Maintainability are viewed in what may be a new light for some. The paper explains why Software Engineers must consider each of these topics as functions of the deliverable system.

The purpose of the paper is two-fold: (1) To give the Computer Science or Engineering student some insight into the application world; (2) To give the individual involved in systems development or maintenance some alternatives regarding that 70%± of the Systems Life Cycle cost.

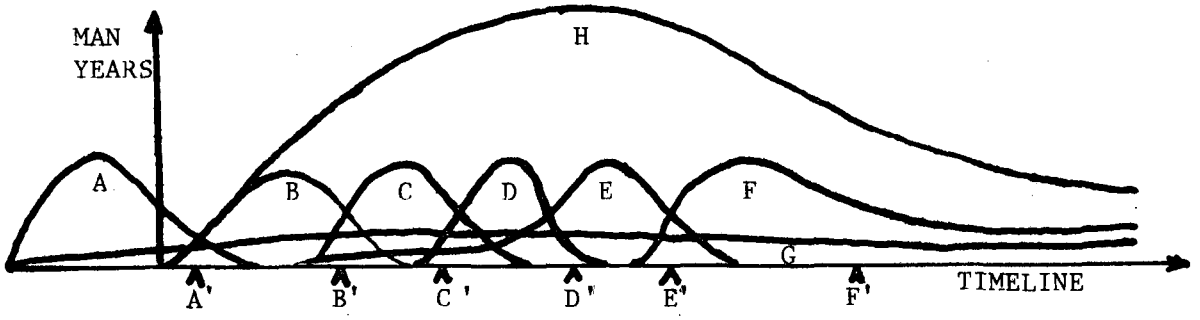
1. *Background*

After working in several different organizations over the past eleven years, it became commonplace to refer to maintenance as "fighting fires" or "working in the trenches" as if we were at war with the user. The somewhat haphazard or unscientific ways of developing and maintaining automated data systems was a way of life. I wasn't really taught to document my code. Even if I had been schooled on the subject, documenting would have come after strong protest.

After being promoted to a systems analyst position, I saw a need to communicate with something besides the computer and sometimes my wife. In most cases, however, the system which I developed I also maintained. The result was that even at this point documenting and representing the system on paper came slowly to me. Only after being transferred to a function which I knew little about and being required to bring to operational readiness a set of half programs with no legible documentation did I start to understand the problem.

Structured Programming and Structured Design are great tools as far as they go; however, they are of little help in the conceptual and analysis phases of the Systems Life Cycle (Figure 1, line A & B). At last year's Software Engineering Conference, a number of case studies pointed out the tremendous cost of maintaining ill defined systems. For example, one case study pointed out that correcting a .5 manday design error in the early stages (Figure 1, point A' & B') would cost from 20 to 50 mandays in operation (Figure 1, line F). This was substantiated by two other studies resulting in somewhat higher cost comparisons. Of equal importance and contrary to common belief was the "discovery" that most program changes were a result of systems design errors and not programming errors. Examples of fault correction ranged from a 70%-30% mix to a 60%-40% mix in two studies (Ref IEEE-76-1). Indeed, our early quantitative description of systems capabilities is mandatory. The documentation should include how and under what circumstances a decision was made as well as the what and why (Ref IEEE-76-2).

FIGURE 1



A=Concept; B=Analysis; C=Design; D=Programming; E=Quality Control; F=Maintenance; G=Management & Audit; H=Development Curve
A'=Concept Review; B'=Analysis Rev; C'=Design Review; D'Internal Delivery Review; E'=Customer Acceptance; F'=Field Audit

(Ref IEEE-76-3)

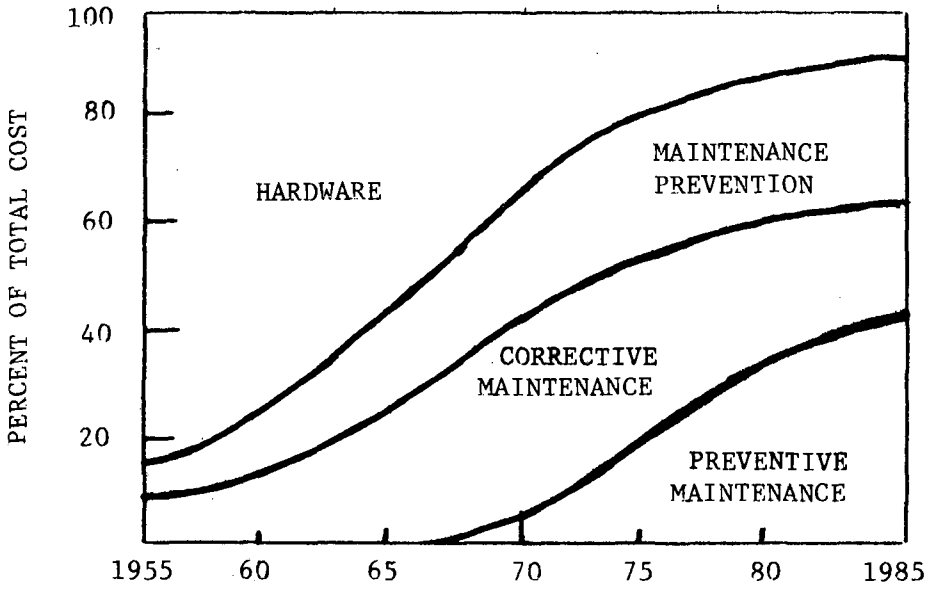
R. L. Patrick (Ref Data-76-1) explains that many people look at costs from the "design", "code", and "test" activities with the old 40%-20%-40% ideas. However, in real life the charts must show "maintenance" as well. This cuts the pie chart into 70%-12%-6%-12% proportions. I believe that we must look at the Systems Life Cycle from the "new" angle. That specifically is looking from the Maintenance point of view.

2. *Maintenance Policy*

In researching the problem of Maintenance, I found that W. M. Lindhorst (Ref Data-73-1) got the idea of scheduled maintenance. J. W. Mooney (Ref Data-75-1) has expanded the topic to include "repair", "revisions" and "enhancements". We have a need to perform a further abstraction on the subject. To do that I studied our friends in heavy industry.

Maintenance Engineering has become recognized in industry as a profession since about 1950. I assert that we should use their many years of experience in resolving our problem. They view Maintenance as being a three-pronged problem including "Preventive Maintenance", "Corrective Maintenance", and "Maintenance Prevention" (Ref BK-1). Using B. W. Boehm's cost trends chart (Ref IEEE-76-2) and information from case studies, our 1985 software cost goals should reflect 30% Maintenance Prevention, 47% Preventive Maintenance, and 23% Corrective Maintenance (Ref Fig 2).

FIGURE 2



3. *Maintenance Prevention*

Maintenance Prevention is simply generating the correct system in the first place. It requires the proper degree of abstraction in the early phases (Fig 1, Line A & B). Maintenance prevention requires modularity in systems design as well as the programming phase (Fig 1, Line C & D). Environmental systems testing of the programs, users and operations manuals

is critical (Figure 1, Line E). However, the prime factor in Maintenance Prevention is proper management (Figure 1, Line G).

3.1. *Proper Management*

Proper management starts at the top with the Vice President for Data Processing. This individual leavies overall (long range) goals regarding data automation concepts, objectives, policies and plans. The individual systems managers must insure that his short range goals and plans are in accord with the corporate goals. Otherwise his system is doomed ultimately to failure.

We have three basic types of plans which must be generated for good Maintenance Prevention. They are the Configuration Management Plan, the Data Project Plans and Resource Plans. All planning must follow corporate guidelines; however, all three plans must be generated in a bottom-up form. The Software Manager, or line management generates information for the next higher level and that level for the next level. Ultimately all plans are consolidated by the appropriate management office to be included at the macro level in overall corporate plans. The resource plan includes projecting both personnel and hardware acquisition. However, those activities are not addressed in this paper.

The Configuration Management Plan should have basic outlines or goals levied by the Vice President for Data Processing. However, for the Configuration Management Plan to operate effectively, it must be generated by the Software Manager for the specific system in question. The consolidated Configuration Management Plan for one set of hardware would insure that all systems plans are compatible and would result in easier control by top management.

Software Data Project Plans should be generated/updated at each review "check point" within each phase of development activity (Ref Figure 1, Point A' B' C' D' E' F'). That is, the Data Project plans would be developed at the start of any project and reviewed/updated at the termination of each phase. Also, a plan would have to be generated for each individual phase.

3.2. *Software Engineering*

Good Engineering needs up-to-date tools and techniques. Examples are SADT, PSA/PSL, PARNAS, the Jackson Design Method, TRW's BMD System, Structured Design, Structured Programming and the Chief Programmer Team (Ref IEEE-76-2). Francis Kelly the noted art restorer summed it up well for his profession and his words apply here also.

> *"Every work starts off on a progressive path to destruction from the moment it is created. One can only hope to delay this time as long as possible by the judicious choice of materials and their application with a sound technique."*

The professional art restorer and the Software Engineer interested in maintenance have numerous simularities. (1) Both are charged with the continuation of a specific item. (2) Both use scientific methods and tools to filter out the problems and in the correction of problems. (3) Both work to prevent the items destruction in time. (4) Both have goals of making the deliverable

as friendly to the end user as possible. The restorer, however, is charged with bringing the work of art back to the original condition. Conversely, the Software Engineer is charged with insuring that the system is not brought back to its original condition.

3.3. *The Alternative System.*

Our goal in Software Engineering should be to generate a dynamic system with: a high maintainability ratio, several capability levels, and measurable availability/reliability. The alternative system should be more than just friendly to developer, user and maintainer. It should be convivial. Joseph E. Worcester's New Dictionary of 1888 defines convivial as follows:

> *"Convivial (kon-v̆iv'e-al) SYN.--The leading idea of Convivial is that of sensual indulgence, festivity, or the pleasures of the table, that of social, the enjoyment from an intercourse with society, festive or jovial company."*

Yes, our alternative to chaos is a friend that makes play out of work. Our friend should be user oriented. Our friend should consider the maintenance team as one of the three users. Our friend must be capable, available, reliable, and maintainable.

The best example of capability is a system which gives you numerous alternatives for performing a task. If, for example, the prompting features of the on-line system went down, then you could fall back to the data name/date type entry. If that set of code failed, then fall-back could go to strict formatting over the remote. If the remote lost power, then entries could be put in at the end of day and beginning of day. If all else fails, the convivial system would play chess with you until Corrective Maintenance was completed.

For our system to be available, it must be operable and in a committable state whenever we need it. Our systems reliability is the probability that a systems configuration item would perform its intended function for a specified period of time (Ref DMJ-75-2, Data 73-2, COMP-74-1).

Our convivial systems Maintainability is directly related to the wisdom of its creator. It should have modular design and structured programming attributes, but more important is its solution to the problem. If the solution is cast in concrete or is in fact a solution to the wrong problem, then it cannot be convivial. The system must be documented and even that documentation must be maintainable (i.e., modular, understandable). Feedback loops must exist between the user, maintainer and developer.

3.4. *Summary of Maintenance Prevention*

Maintenance Prevention is in reality an honest attempt at deleting the other two aspects of maintenance. In short, you try your best to get rid of your own job. This insane act requires a judicious choice of materials and

their sound application. Maintenance Prevention can have the latest in technology and employees who know how to use it, but without proper management the system will fail. Finally, the Software Engineer must look at the Systems Life Cycle with several alternatives in mind. He must consider availability, reliability, capability and maintainability. For other articles on tools, configuration management, Software Management in general and software engineering reference DMJ 75-1, IEEE-76-4, and ACM-72-1.

4. Preventive Maintenance

Preventive Maintenance comes about when the user or maintenance staff realizes that the existing system will soon be in error. This could happen as a result in changed laws or changes in corporate goals requiring a systems upgrade (Ref Data-75-1).

This activity should be conducted as if it were a *mini* systems development. Hopefully the Maintenance Prevention team has: (1) generated sufficient information to ease the reserach time, and (2) created a friend that likes change opposed to the dumb beast that fights tooth and nail. Regardless, the Preventive Maintenance activity must start at concept and walk through each phase, changing affected documentation, configuration items, and programs until the new version of the system is released (Ref Figure 1, Point E').

This type of activity may happen numerous times and in fact several could be going on at the same time, resulting in a single release. Mike Lindhorst in his article on scheduled maintenance gives good advice in this area (Ref Data-73-1).

5. Corrective Maintenance

Corrective Maintenance is usually associated with the crisis or emergency (Ref Data-76-2). It can happen in one of three ways: the system was designed in error, the coding was in error, or Preventive Maintenance was now performed.

The emergency problem must go through a micro development effort just as the other two except that normally it would not alter in-house documentation. Special care must be given to this activity including all tools and techniques available (Ref Data-76-3).

6. Conclusion

We must change lest we bury ourselves in our own creation. We must generate more available and reliable software that performs the same day after day. We must generate maintainable systems that will roll with the punches of changing environment. We must also generate several levels of capability without redundancy. One step in the right direction is viewing the problem from the maintenance point of view. A second step is good old fashioned professionalism in the choice and application of materials, management theory and technology!

References

Books

The Encyclopedia of Management, Van Nostrand Reinhold Co., 1973.

Articles and Periodicals

ACM 72-1, Parnas, D. C. "On the Criteria to be used in Decomposing Systems into Modules", Communications of the ACM, Dec 72.

COMP-74-1, Anderson, P. G., Crandon, L. H. "Computer Program Reliability", Computers and People, July 1974.

DATA-73-1, Lindhorst, W. M. "Scheduled Maintenance of Applications Software", Datamation, Feb 75.

DATA-73-2, Carey, L. J. "IEEE Symposium on Software Reliability", Datamation.

DATA-75-1, Mooney, J. W. "Organized Program Maintenance", Datamation, Feb 72.

DATA-76-1, Patrick, R. L. "Software Engineering and Life Cycle Planning", Datamation, Dec 76.

DATA-76-2, Liu, C. C. "A Look at Software Maintenance", Datamation, Nov 76.

DATA-76-3, Shaw, D. C. "Managing a Software Emergency", Datamation, Nov 76.

DMJ-75-1, Defence Management Journal, Oct 75.

DMJ-72-2, Manley, J. H. "Embedded Computer System Software Reliability", Defense Management Journal, Oct 75.

IEEE-76-1, Proceedings of the 2nd International Conference on Software Engineering, IEEE Catalogue No. 76CH1125-4C.

IEEE-76-2, Tutorial on Software Design Techniques, IEEE Catalogue No. 76CH1145-2C.

IEEE-76-3, Designing with Microprocessors, Fall COMPCON-76.

IEEE-76-4, Hamilton, M. and Zeldin, S. "Higher Order Software--A Methodology for Defining Software", IEEE Transactions on Software Engineering.