Technology transfer in the maintenance environment

by FLORENCE J. BELL
The Equitable Life Assurance Society
of the United States
New York, New York

ABSTRACT

In 1982 The Equitable Life Assurance Society of the United States recognized that software maintenance requires major management attention, and established a maintenance productivity project (MPP). Maintenance was defined as any programming effort that requires at least 25% of a programmer's time to be spent understanding an existing system. Three potential areas were identified for technology transfer: the maintenance function, the maintenance environment, and maintenance metrics. Ongoing programs include cooperation with vendors in developing an integrated environment for the maintenance programmer and manager, a maintenance management handbook, and a maintenance managers' round table. Maintenance is becoming an established and recognized area of specialization for systems professionals at The Equitable.
INTRODUCTION

The Equitable Life Assurance Society of the United States is the third largest mutual life insurance company in the U.S., with assets of more than $45 billion and about $230 billion of life insurance in force. The company installed its first mainframe, an IBM 650, in 1956, and at that time established its systems development department, with a total complement of three people. Twenty-seven years later The Equitable had a total of eight mainframes with over 60 mips capacity, 750 systems professionals, an annual systems budget of $100 million, and an inventory of approximately 350 major systems with 7000 program modules.

In 1974, in keeping with a general decentralization of the company's management, the systems development department was divided into five independent units, whose heads reported to line management. By 1983 there were nine autonomous systems departments. When the systems development department was decentralized, an EDP coordinating committee was formed, composed of the officers who headed each of the systems departments, the head of the data processing department, and the technology officer. The committee was responsible for ensuring that the systems needs of the corporation as a whole were met; specifically that hardware support was available, that well-qualified systems professionals were recruited, trained, and developed, that advances in hardware and software technology and in systems development management were introduced into the company, and that the economies of scale of an EDP installation as large as The Equitable's were not lost through the decentralization.

In 1980 the EDP coordinating committee established an application productivity group (APG) with the charter of technology transfer, specifically to increase the productivity of The Equitable's systems effort by a factor of ten within a period of five years. Within its first two years, the APG introduced interactive computing throughout all systems areas, selected and installed the hardware and operating systems for the interactive testing environment, and established a special interactive testing support organization. The group also introduced the concept of end-user systems development, brought the FOCUS language and database management into the company, and conducted extensive user training.

In 1982, the EDP coordinating committee conducted an off-site planning session to set the direction for future efforts of the APG. At this time, maintenance, methodology, and prototyping were identified as primary areas of concern. Of these, maintenance—which at the beginning of the session had little support—emerged as the top priority, primarily because of an awareness that although maintenance used over half of the systems resources, it had been disregarded in the systems development methodology installed 10 years earlier.

INITIAL SURVEY

Between September and December of 1982, the APG conducted its initial survey of the maintenance effort throughout the company. The purpose of this survey was to define the specific goals of a maintenance productivity project (MPP), to estimate the realizable benefits, and to establish a level of effort and a timetable.

As a first step, the group contracted for the services of Julien Green, a senior consultant with wide systems experience and a thorough knowledge of The Equitable's systems environment. With him, we reviewed current literature and interviewed managers in most of the systems areas to identify the specific needs of The Equitable's maintenance managers and programmers.

The results of this investigation were published in December 1982, and can be summarized under the following headings:

1. Definition of the maintenance function
2. Definition of the maintenance environment
3. Definition of maintenance metrics
4. Project deliverables

Definition of the Maintenance Function

The industry has developed what is now a generally agreed upon terminology in describing maintenance, based upon Swanson's original classification: corrective, adaptive, and perfective maintenance.1 Corrective maintenance is fixing errors. Adaptive maintenance is changing software to accommodate changes in the computing or business environments without affecting the software's function. Perfective maintenance is enhancing function.

These three quite dissimilar activities have in common the requirement that the programmer spend a considerable portion of time (estimated by Fjeldstad and Hamlen at 50%) in understanding existing materials (code, documentation and procedures).2 It is this requirement that distinguishes systems maintenance from systems development.

For the purposes of our MPP we define maintenance as any programming effort that requires at least 25% of the programmer's time to be spent understanding an existing system. We believe this is the point at which programmers begin to benefit from maintenance-specific tools, which facilitate the analysis of systems as opposed to their synthesis. If we were to set this cut-off at a lower percentage, we would include some clearly development-type programming, which in a mature EDP environment such as ours usually requires interfacing with, and therefore understanding, existing systems.
We had reviewed other operational definitions used by systems managers; some distinguish small jobs (maintenance) vs. large ones (development); others distinguish modification of existing code (maintenance) vs. the creation of new modules (development); still others, following Barry Boehm,\(^3\) include redesign of less than 50% of existing code (maintenance) vs. redesign of more than 50% (development). We noted however that some small jobs are free-standing, while some large jobs are large precisely because they involve manipulation (i.e., maintenance) of a large existing system; that some projects that require little or no modification of existing systems nevertheless require a major effort in understanding them; and that the redesign of a larger percentage of an existing system requires a greater maintenance effort than the redesign of a smaller percentage.

Accordingly, we concluded that the level of effort required by a technician to understand an existing system is a more fundamental criterion than others that have been proposed. Furthermore, it appears that an operational definition of maintenance from the systems manager's point of view must factor in the cost of understanding code. From this viewpoint, defining maintenance in terms of the effort required to understand existing code makes sense.

**Definition of the Maintenance Environment**

Our initial survey also identified three components of the maintenance environment, each with its own needs. The first component is the programmers' environment. We found that many tools used in development work were used by maintenance programmers, but that there was a need for tools that addressed the maintenance-specific function of understanding existing code. We also found that, although there were useful maintenance tools, no single product purported to provide an integrated environment—a situation quite different from that on the development side of the house, where it has long been recognized that the greatest productivity gains come not from the sum of the tools, but from the integration of the tools into a structured environment.

The second component of the maintenance environment is the managers' environment. Here we found a need for management tools—packages to assist in estimating programming effort, scheduling and controlling maintenance work, budgeting, and reporting. Again some tools used for development were useful, but some, such as an effort estimator for maintenance work, were not available. In addition there was a need for a description of the sequential steps in maintenance work, and for a checklist with which to determine the accomplishment of each step.

The third component of the maintenance environment is the institutional environment, which encompasses the issues of the image of maintenance, selection and training of maintenance personnel, and career paths for maintenance professionals.

**Definition of Maintenance Metrics**

Finally, the initial survey identified the need for a good set of maintenance metrics upon which to base rational maintenance decisions. Two types of metrics are needed: First are macro-metrics—used to provide a multidimensional profile of our software inventory. These metrics will allow us to estimate the size, complexity and state of deterioration (or health) of our existing software portfolio, predict the resources needed to maintain our inventory, estimate the cost of maintenance, and identify areas of largest payoff. An example of a macro-metric is the number of man-months required to maintain the "average" program module.

Second are the micro-metrics—used to provide information needed for decisions concerning the maintenance of individual systems. These metrics will serve as a basis for determining when to retire, restructure, or retrofit a system, for measuring productivity trends, for estimating the time and cost of specific maintenance jobs, for preparing an annual maintenance budget, and for evaluating proposed new software tools. An example of a micro-metric is an algorithm to estimate the man-months required to implement a specific program enhancement.

**Project Deliverables**

Maintenance improvement is an unusually difficult environment for technology transfer. Installed systems cannot be easily adjusted to use a predefined tool or component; nor can an abrupt change of method be implemented by a staff carrying a full load of projects already in progress. A maintenance productivity project does not consist of installing tools, or adopting a methodology, or establishing management policies. Instead, it requires continuing action on several levels.

Therefore, the initial survey defined our objective as introducing technology transfer into an integrated maintenance environment upon a foundation of sound maintenance metrics. A set of project deliverables for each component of the environment was developed.

These included, for the programmers' environment, a maintenance workbench, i.e., a set of software tools integrated through a common gateway or front end.

Project deliverables specified for the managers' environment were a handbook containing an inventory of the tools in the maintenance workbench, with guidelines for their appropriate use, costs, and expected benefits, a description of the maintenance process, and a milestone checklist; and a set of software tools, probably resident on a personal computer, for estimating, scheduling, controlling, and budgeting maintenance work.

Finally, for the institutional environment, a maintenance managers' round table was recommended. This is a periodic meeting of systems managers to define common maintenance concerns, exchange successful solutions, and channel technical advice. The round table is designed to build a community of interest and to be the main line of communication for technology transfer, for evaluating and integrating tools, for drafting the handbook, and for originating new avenues of investigation.

**Strategy**

In November 1982, the Application Productivity Group began to address the programmer's environment. There were many
reasons why we chose to begin our maintenance project with this activity.

Evaluating and installing software tools is the easiest task for us to work at. Tools pre-exist our efforts, are concrete, and demonstrate measurable results. The APG has had considerable experience in finding, piloting, and evaluating software. Good results are readily realizable through the installation of these tools. Therefore, although we believe that in the long run activities other than the installation and even the integration of tools will prove more important, we started our implementation effort by identifying and evaluating maintenance tools.

Seven types of software tools for the Maintenance Workbench were identified for further investigation. They were retrofitters, restructurers, static code analyzers, interactive debuggers, test data generators, automated documentors, and specialized editors. From among these, we selected a new interactive code analyzer to evaluate and pilot.

INTERACTIVE STATIC ANALYZER BETA TEST

James Martin and Carma McClure had written that “the tool the maintainer most needs is an interactive code analyzer that will help him to understand how the code works, and to predict the side-effects of modification.” At the time we completed our initial survey, a vendor was preparing to beta-test an interactive analyzer.

The APG’s preliminary evaluations at the vendor’s site indicated that the product had powerful functionality. On the basis of this evaluation, The Equitable agreed in February 1983 to be a beta site.

The product loaded COBOL source code to an on-line database, which a maintenance programmer could then access interactively. It presented three views of the program: the structure chart view, which gave the programmer an overview of the design of the program; a source code view, which allowed a programmer to look at selected units of code; and a source code difference view, which presented different versions of the program. In each of these views the program could select and trace data flows and control logic. It was at the time the only interactive static analyzer that we were able to find.

Objectives of the Beta Test

The objectives of the beta test were to:

1. Confirm the functionality of the product. Would it effectively trace the logic and data flows of actual production systems, provide accurate flow charts, and compare differences in source code?
2. Determine the quality of the product. How many bugs would be encountered during the beta test, and how seriously would they affect the product’s functionality?
3. Evaluate the usefulness of the product in a production environment. Would it provide answers to real maintenance questions, and information actually needed to modify programs?
4. Ascertained training requirements. How long would it take programmers to learn to use the product?
5. Determine the practicality of using the product with programs written for the non-IBM-compatible systems. Could minicomputer programs be analyzed?
6. Evaluate the acceptance of the product by The Equitable’s maintenance professionals. If installed, would the product become the systems community’s Edsel?
7. Evaluate the support given by the vendor during the beta test. What level of support might we expect when the product was released?
8. Evaluate the system resources required by the product. What effect would it have on our data centers?
9. Evaluate the system resources required by the product. Would it incur for the use of the product. What would it cost to analyze code with it?
10. Evaluate the actual productivity gains that could be expected. Would benefits outweigh costs?

Results of the Beta Test

The beta test ran from Feb. 2 through April 15, 1983. During the course of the beta test 100 program modules were analyzed, and approximately 250 hours of interactive testing were logged.

At its conclusion, the functionality of the static analyzer was confirmed. On all other factors, except quality, the product received an acceptable or better rating (Figure 1). However, the vendor withdrew the package.

We learned three major lessons from this experience: First, an interactive static analyzer is a valuable tool, and will be well received by programmers. Since the beta test, whenever programmers evaluate a software tool, they invariably compare it to the analyzer and begin their evaluations, “Well, it isn’t a (product), but...” We found that a static analyzer can reduce the time a programmer spends understanding code by 20–50%. In our environment a 23% reduction in programmer time for this function would have offset the machine charges. We look forward to the day when a viable interactive static analyzer is on the market.

<table>
<thead>
<tr>
<th>FACTOR</th>
<th>RATING</th>
</tr>
</thead>
<tbody>
<tr>
<td>Functionality</td>
<td>5</td>
</tr>
<tr>
<td>Quality</td>
<td>4</td>
</tr>
<tr>
<td>Usefulness</td>
<td>3</td>
</tr>
<tr>
<td>Training Requirements</td>
<td>2</td>
</tr>
<tr>
<td>Mini-computer programs</td>
<td>1</td>
</tr>
<tr>
<td>Programmer Acceptance</td>
<td>1</td>
</tr>
<tr>
<td>Vendor Support</td>
<td>1</td>
</tr>
<tr>
<td>Resource Requirements</td>
<td>1</td>
</tr>
<tr>
<td>Running Costs</td>
<td>1</td>
</tr>
<tr>
<td>Productivity Gains</td>
<td>1</td>
</tr>
</tbody>
</table>


Figure 1—Evaluation of interactive static analyzer
Second, we learned more about evaluating maintenance tools. Although most of our criteria had been defined before the test, others emerged during the weekly review meetings we held with the programmers. It was at these meetings that the distinction between functionality, quality, and usefulness was hammered out. We will evaluate other tools against these criteria, as well as against additional criteria that may apply. We expect other maintenance products to appear on the market in the near future, and we intend to integrate the best of them into our environment.

Third, we conclude that the maintenance workbench is a facility whose time has come. The productivity improvement realized by having static analysis functions available in an interactive harness demonstrated the potential benefits of putting many other maintenance functions in such a harness.

CONTINUING ACTIVITIES

At the time of this writing, The Equitable's maintenance productivity improvement program is progressing along the lines laid out in the initial survey. For the programmers' environment, maintenance tools continue to be evaluated. We are particularly looking at packages that restructure and redocument existing code.

For the managers' environment, a maintenance effort estimator has been developed by another consultant to the project, Howard Rubin, as a component of the ESTIMACS package. The maintenance management handbook is being outlined by Julien Green. For the institutional environment, Nicholas Zvegintzov is working with us as a consultant to coordinate the initial meetings of the maintenance managers' round table.

A new software metrics project has been established. Its team will develop the metrics for maintenance specified by the maintenance productivity project, as well as software development measurements.

CONCLUSION

Software maintenance has been a major systems function at The Equitable for many years. It is now recognized as a function whose contribution to the systems and corporate effort deserves the serious attention of upper management. A maintenance productivity improvement program has been developed, approved, and funded. Maintenance is becoming an established and recognized area of specialization for systems professionals at The Equitable.

REFERENCES