59-61
136139
N93-17176
P. 14

Toward Full Life Cycle Control: Adding Maintenance Measurement to the SEL

H. Dieter Rombach and Bradford T. Ulery
Computer Science Department and Umiacs, University of Maryland, College Park, Maryland

Jon D. Valett
NASA, Goddard Space Flight Center, Greenbelt, Maryland

Organization-wide measurement of software products and processes is needed to establish full life cycle control over software products. The Software Engineering Laboratory (SEL)—a joint venture between NASA's Goddard Space Flight Center, the University of Maryland, and Computer Sciences Corporation—started measurement of software development more than 15 years ago. Recently, the measurement of maintenance has been added to the scope of the SEL. In this article, the maintenance measurement program is presented as an addition to the already existing and well-established SEL development measurement program and evaluated in terms of its immediate benefits and long-term improvement potential. Immediate benefits of this program for the SEL include an increased understanding of the maintenance domain, the differences and commonalities between development and maintenance, and the cause-effect relationships between development and maintenance. Initial results from a sample maintenance study are presented to substantiate these benefits. The long-term potential of this program includes the use of maintenance baselines to better plan and manage future projects and to improve development and maintenance practices for future projects wherever warranted.

1. INTRODUCTION

Most software organizations lack satisfactory control over their development and maintenance projects. This lack of control is exemplified by the absence of explicit models enabling the identification of ambiguous product requirements, the selection of practices best suited to achieve given requirements, or the prediction of the impact early project decisions may have on the quality of the resulting products. Each organization has its own set of control problems and reasons standing in the way of improvement. Comprehensive measurement programs are needed as a first step toward improvement [1]. Such programs can help identify the specific problems of an organization in quantitative terms, pinpoint possible causes, motivate improvements, and assess alternatives considered for improvement.

The Software Engineering Laboratory (SEL)—a joint venture including government, industry, and university— began measurement of satellite ground support software development projects in 1976. The three primary organizational members of the SEL are the Systems Development Branch at NASA's Goddard Space Flight Center, the Computer Science Department at the University of Maryland, and the Systems Development Operation at Computer Sciences Corporation. This collaboration has produced numerous case studies and controlled experiments [2-6]. Results from these case studies and experiments motivated several improvements within the SEL [7-9].

In 1988, the SEL incorporated maintenance into its scope of measurement. The result is an even more comprehensive measurement program in which data is now being collected during development and maintenance of all software systems. In the SEL, pre- and postlaunch maintenance activities are performed by separate organizational entities. Currently, maintenance data are only collected from prelaunch maintenance activities. In the remainder of this article, the term "maintenance" shall refer to this prelaunch phase

Address correspondence to Professor H. Dieter Rombach, AG S ‡ W Eng., Fachbereich Informatik, Universitaet Kaiserslautern, Postfach 3049, D-6750 Kaiserslautern, Germany.

between delivery of a completed software system and the actual launch of the related spacecraft. This maintenance measurement program is customized to the pertinent SEL characteristics, including the definition of maintenance, the maintenance improvement goals, and other product, process, and people factors.

Empirical research in the SEL is based on the idea of continuous improvement. This idea has been formulated as the quality improvement paradigm [1]. According to this paradigm, improvement is the result of continuously understanding current practices, changing them, and empirically validating the impact of these changes. Improvement requires measurement.

In the SEL, measurement goals define the data to be collected and provide the context for data interpretation. This goal-oriented approach to measurement has been formulated as the goal/question/metric paradigm [1, 10, 11]. It suggests defining each goal by developing a set of analysis questions, which in turn lead to a set of metrics and data. The short-term goals of our maintenance measurement program have been to increase the understanding of maintenance within the SEL; the long-term goals are to stimulate improvements in the SEL's ability to plan and manage future maintenance projects and—whenever needed—to motivate the use of different development and maintenance practices.

Specific characteristics of the SEL maintenance environment as well as the comprehensive scope of our measurement approach make this program unique. The study results presented here may not be directly comparable to those from other maintenance environments, yet they do show how a comprehensive measure program can be used to better understand and improve an organization's development and maintenance process and products. Few comprehensive maintenance studies have been published [12-14]. Most empirical maintenance studies report on laboratory-style controlled experiments [15, 16], isolated case studies [13, 17], or project surveys [18]. A survey of maintenance studies has been published by Hale and Haworth [19].

The purpose of this article is to state our initial maintenance study goals and questions, present the related results, and propose—based on what we have learned—a revised set of goals and questions for future studies.

The study results are organized according to the types of data used to address the goals and questions: quantitative maintenance baselines, comparisons between quantitative development and maintenance baselines, and qualitative information regarding the cause-effect relationships between development and maintenance. These results have increased our understanding of maintenance processes and maintained products in the SEL, commonalities and differences between development and maintenance, and development characteristics affecting maintenance. On occasion, our results are carefully compared with results from other published studies or widely believed maintenance myths.

We begin our presentation with a background discussion of the SEL and the new maintenance measurement program (sections 2 and 3, respectively). We then present the results of our study (section 4). We conclude with an assessment of the SEL maintenance measurement program and a revised set of goals and questions for future maintenance studies.

2. THE SEL

The goals of the SEL are to understand its software development processes, to measure the effects of various methods and tools on these processes, and to identify and then apply new, improved development practices. Improved understanding within this particular environment provides the basis for better planning and management as well as a rationale for adopting new practices [4].

Development in the SEL supports satellite missions. SEL studies generally focus on attitude ground support systems and their associated simulators. These product lines are very stable: the system architecture, documentation standards, and organizational responsibilities do not change significantly from one mission to another. Attitude ground support systems have 130-240K lines of FORTRAN source code (where a line of code is measured as a physical line, including comment lines) and require 15-30 staff years to develop. Simulators have 25-75K lines and require 3-10 staff years to develop.

Research in this environment is guided by two basic paradigms: the quality improvement paradigm (QIP) and the goal/question/metric paradigm (GQM). The QIP, which applies the principle of continuous improvement to software engineering, defines the context for measurement within the SEL [1]. Accordingly, software development can be improved by iterating the following steps for each project: (1) characterize the corporate environment; (2) state improvement goals in quantitative terms; (3) plan the appropriate development practices and methodologies together with measurement procedures for the project at hand; (4) perform the development and measure, analyze, and provide feedback; and (5) perform postmortem analysis and provide recommendations for future projects. Each QIP iteration is characterized by its own set of goals. These goals reflect—and evolve with—the maturity of the investigated organization.

Measurement in the SEL is guided by the GQM paradigm [10]. Measurement is used to characterize current development practices, monitor and manage development projects, identify strengths and weaknesses of the current practices, and evaluate promising new technologies in a controlled environment. The GQM paradigm describes a goal-oriented approach to measurement in which metrics are tied to specific measurement goals. According to the GQM paradigm, each measurement goal is listed explicitly, a set of specific questions is posed to address each goal, and specific metrics and measurement procedures are defined to support the questions. The resulting data collection procedures and interpretations are tailormade to the study's goals and local environment characteristics. For instance, in the SEL, this generally means that metrics and measurement procedures reflect the use of SEL-specific development practices, fit the organizational structure, and permit comparisons with historical data. Goals, questions, and metrics provide a context that helps ensure that data are interpreted correctly and are compared only to data and results from similar contexts.

Two types of measurement are common in the SEL: routine monitoring and exploratory studies. Routine monitoring is used to characterize the local environment broadly. The resulting quantitative and qualitative baselines are used to plan and manage new projects and to compare the effects of newly introduced tools or methods against [6]. Objective and subjective data are routinely gathered for each project [20]. Objective data include staff hours, computer utilization, source code growth, and the number and kinds of changes made to the source code. Subjective data characterize the software development process and software product characteristics. The data for over 100 projects monitored over the last 15 years is maintained in the SEL database [21].

Exploratory studies are used when the SEL is in the initial phase of understanding a process or methodology. For example, the SEL is currently studying three projects following the cleanroom methodology [22]. Special data collection procedures were designed for these projects to permit researchers to monitor the effort spent in reading and reviewing designs and code.

Measurement in the SEL has provided a rationale for making evolutionary changes to NASA's development practices, including stricter use of code-reading techniques [5], guidelines for Ada projects [23], and the adoption of the cleanroom development approach [24]. With the addition of maintenance measurement, the SEL is attempting to lay the foundation for similar improvements in maintenance.

3. THE SEL MAINTENANCE MEASUREMENT PROGRAM

The following subsections describe the SEL maintenance environment and the specific goals and procedures of our measurement program. A more detailed description of this environment, its products, and maintenance processes appeared in the proceedings of the 1989 IEEE Conference on Software Maintenance [25].

3.1 Maintenance Environment

In the SEL, maintenance is partly defined by organizational responsibility and schedule. As depicted in Figure 1, each product passes through three different organizational units during its lifetime: analysts produce the initial functional specifications used by the developers and remain responsible for these specifications throughout development and until launch; operations assumes complete responsibility after launch. During the period between development and launch, the analysts have complete responsibility for the system, including the implementation of any changes.

In this study, maintenance refers specifically to software change activities performed by the analysts during the postdevelopment, prelaunch phase. By nature of these constraints, the maintenance phase is typically shorter in the SEL than in other environments (one to two years), and the maintenance changes are not triggered by operational failures but by failures detected during simulated uses of the software by prospective operators and externally triggered changes of the overall satellite mission.

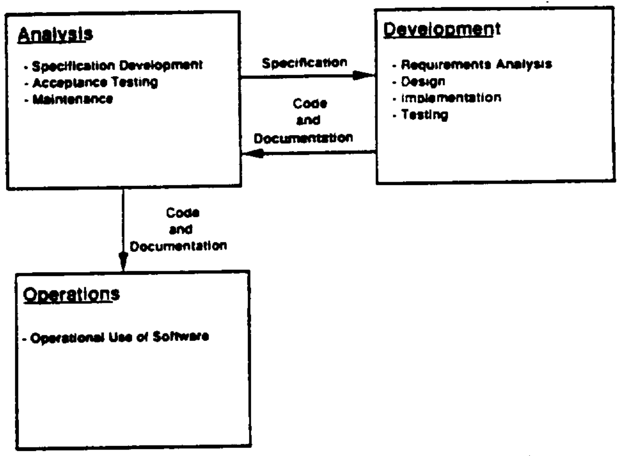


Figure 1. Organizational structure of the SEL environment.

The products maintained are the same simulators and attitude ground support systems described in section 2. Typically, the effort expended during the one- to two-year time frame that these systems are in maintenance is approximately 5% of the development effort. Maintenance procedures vary from project to project depending on the type of system being maintained, the size of the maintenance team (2-10 people on the projects studied), the specific methods and tools elected by the individual programmers, and other factors. In general, formal change control procedures are followed; changes are implemented one at a time, but may be tested in groups; and one maintainer is responsible for implementing each change.

3.2 Maintenance Measurement Goals

Consistent with the overall directions of the SEL, we chose three general goals for the maintenance measurement program: (1) to understand maintenance processes and products better; (2) to improve our ability to manage current maintenance projects and plan future ones; and (3) to establish a sound basis for improving development from a maintenance perspective.

Following the QIP, the initial goals focus on understanding maintenance. Representative measurement goals and questions selected for this study are summarized in Figures 2, 5, and 11. Analysis results related to these goals and questions are presented in section 4.

3.3 Maintenance Measurement Procedures

The data collection procedures used in this study were designed according to the principles of the GQM paradigm. Data were collected via exploratory interviews and routine data collection forms [20]. The routine data collection forms used during maintenance include the Weekly Maintenance Effort Form and the Maintenance Change Report Form (Appendix A). The effort form is filled out once per week per maintainer per system; one change form is filled out per completed change. The weekly effort forms record the distribution of effort (in staff hours) by type of change (correction, enhancement, adaptation, or other[1]) and by engineering activity (designing, coding, etc.). The change forms record the distribution of changes by type of change, size of change, changed objects (e.g., code, user's guide), expended staff time, fault type (if applicable), and more. All data are validated through a series of checks by the data entry personnel, project managers, and SEL researchers. Data are stored and made available to researchers and developers through the SEL database [21].

[1]All maintenance effort that cannot be attributed to an individual maintenance change is classified as "other." This includes effort related to management, meetings, and training.

4. MAINTENANCE MEASUREMENT BENEFITS

The maintenance measurement program has already increased understanding of maintenance in the SEL. Previously, much of this understanding was at best intuitive and approximate. In this section we demonstrate what we have learned as a result of our initial study. The results are separated into baseline characterizations of maintenance, a comparative analysis of development and maintenance, and an analysis of how development decisions affect maintenance.

In this study, we restrict our analyses to three large attitude ground support systems for which we have complete and valid data: the Gamma Ray Observatory, the Geostationary Operational Environmental Satellite, and the Cosmic Background Explorer. Maintenance of these systems was performed between 1988 and 1991. A total of 90 changes and over 10,000 hours of effort serve as the basis for all quantitative analyses of maintenance presented here.

Examining the data on these three projects has provided valuable insight into the maintenance process within this environment. The results presented here are intended to demonstrate the increased understanding of the maintenance process that can result from a measurement program.

4.1 Maintenance Baselines

The first step toward understanding any environment is to develop baselines describing that environment [12, 14]. The goals and questions related to this part of the SEL study are listed in Figure 2. They are intended

GOAL 1: Characterize the changes performed during maintenance.
- QUESTION 1
  - How many changes of each type are completed?
- QUESTION 2
  - How much effort is spent on changes of each type?

GOAL 2: Characterize product evolution during maintenance.
- QUESTION 3
  - How much code is affected by each change?
- QUESTION 4
  - Is code added, changed or deleted?

GOAL 3: Characterize the maintenance process stability
- QUESTION 5
  - How do maintenance processes differ across projects?

Figure 2. Measurement goals for understanding maintenance in the SEL.

to characterize what kinds of changes are performed during maintenance, which parts of the systems change and how, and what maintenance processes are followed. In the long term, the resulting baselines are expected to provide a basis for determining whether new techniques or process adjustments have any measurable impact on the SEL maintenance processes or products. Any comparison between SEL baselines and baselines from other environments must take environmental differences into account.

Each maintenance change in this environment is well defined by a formal change request. There are several key steps in the change process: changes must be approved, implemented, tested, and released. In general, more changes are approved than can be implemented. This poses the difficult management problem of selecting which changes to implement. This decision is based on the importance of the changes approved as well as the budget available to make changes. The implementation of a change is performed by one programmer; there is no standard, formal methodology. Testing, beyond debugging by the programmer, is performed for several changes at once. One important implication is that the associated effort measured cannot be ascribed to a particular change. In fact, testing is typically performed at two levels: the first level provides internal checkpoints for configuration management; the second level occurs before each release.

Each maintenance change performed in the SEL is classified as an enhancement, adaptation, or correction [26]. A simple count of changes suggests that maintenance is primarily corrective; however, the effort distribution reveals that most effort is actually related to enhancements (Figure 3). Either way, adaptations do not seem to contribute significantly (Figure 2, questions 1 and 2). Note that the average enhancement requires just over twice the effort of the average correction. This phenomenon could be caused by the fact that enhancements are typically larger than corrections, that enhancements are inherently more difficult to accommodate into an existing system, or both.

As early as 1976, Belady and Lehman [14] demonstrated the benefits of program evolution models for the purpose of understanding the decay of software undergoing change. Figure 4 summarizes how many modules and lines of source code have been added, changed, or deleted per change (Figure 2, questions 3 and 4). On average, three lines of code are added for every existing line changed or deleted. Entire modules are rarely added and never deleted. In the SEL, maintainers do not significantly alter the system's architecture to make changes. We hypothesize that the high number of lines added reflects the high proportion of enhancements, and that architectural stability reflects an "if it ain't broke don't fix it" attitude. Such an attitude could be explained by the general lack of understanding of overall system architecture. The observed growth pattern also suggests that module functionality increases during maintenance, leading to a decrease in module cohesion. Decreased cohesion may not be a problem during the short lifespan of a satellite system, but may reduce the reuse potential of modules in future developments.

Our most striking observation about SEL maintenance is the extent to which the maintenance processes vary across similar projects (Figure 2, question 5). Some of the variability reflects the size and composition of the maintenance teams (2–10 programmers). One particular area where the processes differ appears to be in the approach to testing. The projects studied have not established well-defined criteria for when system or integration testing should be performed during maintenance. Such variability in the process reflects the relatively ad hoc nature of the maintenance environment as compared to the development environment. In fact,

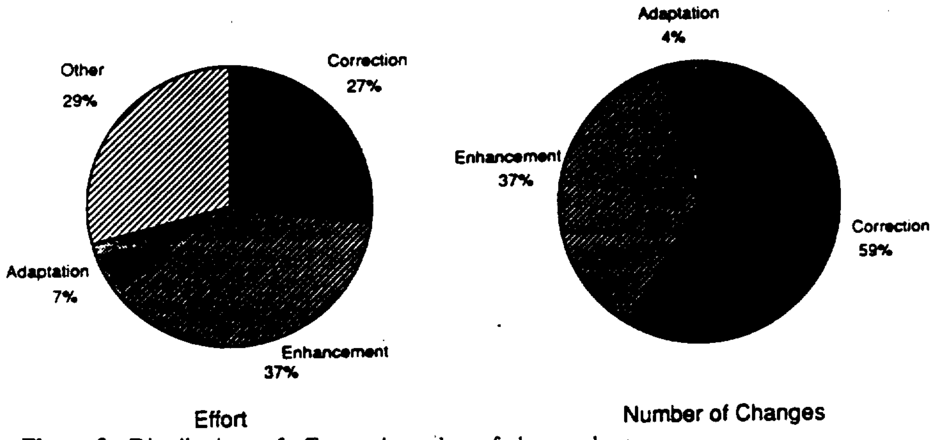


Figure 3. Distributions of effort and number of changes by type.

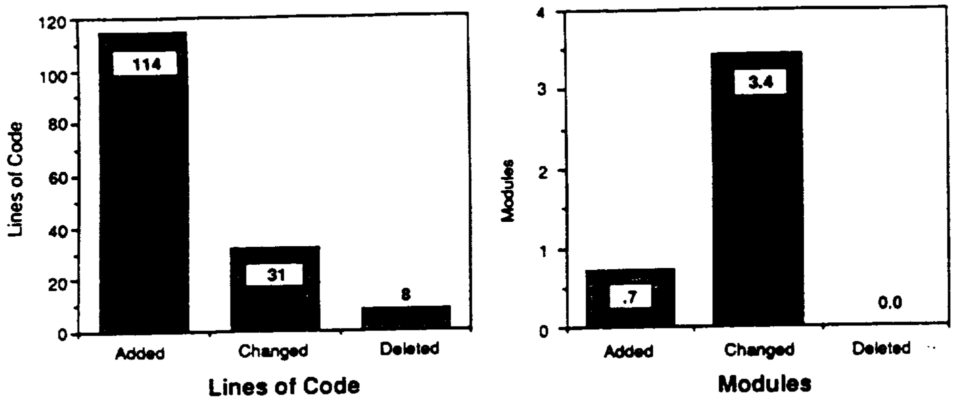


Figure 4. Lines of code and modules per change.

studies such as this one aim at increasing the maturity of the maintenance process within this environment. By identifying which aspects of the process are most successful, a single consistent process will be identified.

4.2 Maintenance vs. Development

Applying experience from past development studies to maintenance requires an understanding of the similarities and differences between maintenance and development. The goals of this part of the study were to compare changes made during development and maintenance, types of changes, and change processes (Figure 5). These comparisons are possible because both development and maintenance data are available for the three systems studied.

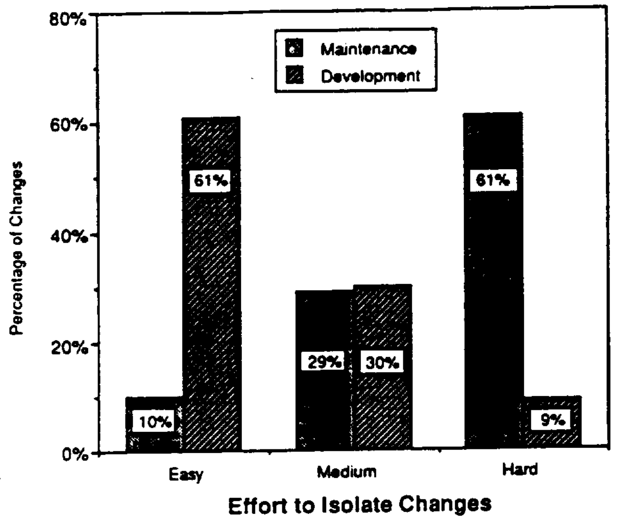
Throughout development and maintenance, the effort spent on each change is recorded. Effort is classified as easy when it takes less than an hour to complete a change, medium when it takes between an hour and a day, and hard otherwise. A distinction is made between the effort to isolate a change (understand the request and locate the affected modules) and the effort to complete the change (design, code, test). Figure 6 shows that changes performed during maintenance generally require more effort than those performed during development (Figure 5, question 6). We consider two hypotheses that might account for this pattern: changes requested during maintenance are inherently harder than those requested during development; and it is more difficult to perform the same change during maintenance than it would be during development. While we cannot determine whether particular modules are easy or difficult to change during maintenance based on our data, we are able to examine both hypotheses further at the level of the individual change.

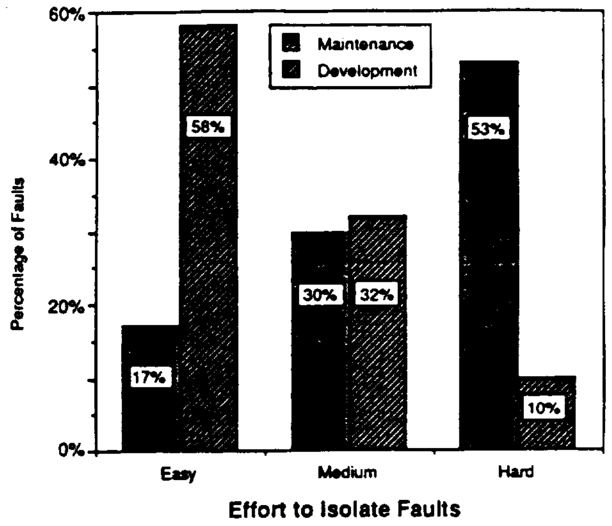
Regarding the first hypothesis, we find no obvious difference between the effort distribution patterns for all changes (Figure 6) and corrections only (Figure 7). We conclude that the increased effort is not primarily due to differences in the distributions of types of changes requested.

Regarding the second hypothesis, various characteristic differences between development and maintenance are commonly thought to explain why the same change might be more difficult to perform during maintenance. These include product factors (such as increased complexity and missing or out-of-date documentation), process factors (such as schedule constraints, methods, and tools), and people factors (such as a lack of familiarity with the software). In the SEL, we cannot attribute the maintenance difficulties to product factors because there is already a sharp increase in change effort during acceptance test, but little change in the products. Instead, we suspect some combination of process and people factors. Although we are unaware of any significant methodological differences between the

GOAL 4: Compare changes made during development and maintenance.
QUESTION 6
How does the effort per change compare?
GOAL 5: Compare the types of changes made to products at both phases.
QUESTION 7
Are the faults found during maintenance different than those found during development?
QUESTION 8
How do the distributions of errors by class compare?
GOAL 6: Compare change processes at both phases.
QUESTION 9
How does the distribution of effort by activity type compare?

Figure 5. Measurement goals for understanding the similarities and differences between development and maintenance.





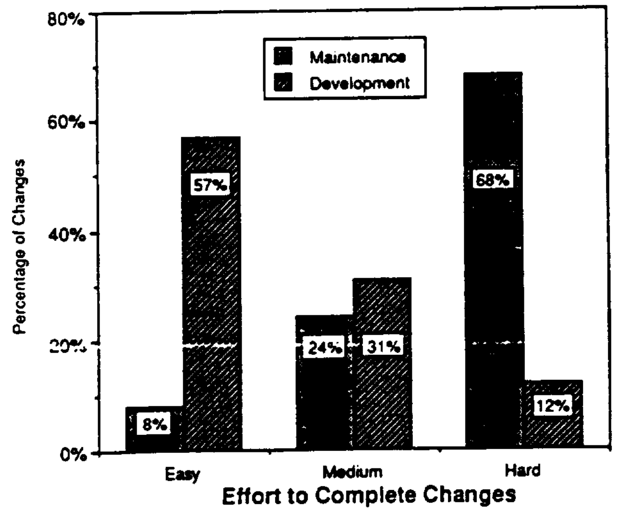


Figure 6. Effort to isolate and complete changes: maintenance vs. development. Easy, ⟨1 hour; medium, ⟩1 hour and ⟨1 day; hard, ⟩1 day.

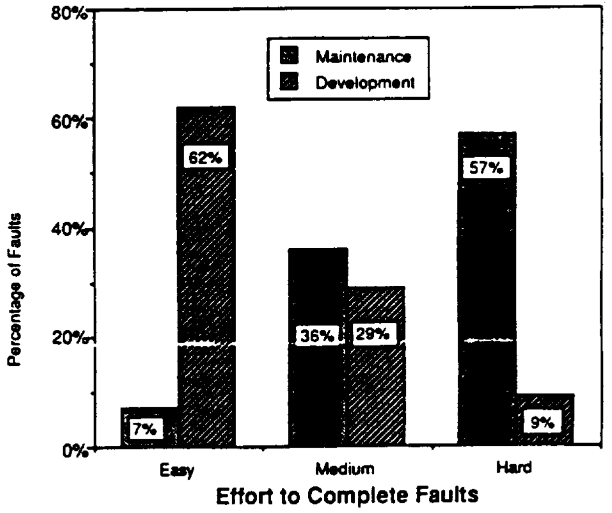


Figure 7. Effort to isolate and complete faults: maintenance vs. development. Easy, ⟨1 hour; medium, ⟩1 hour and ⟨1 day; hard, ⟩1 day.

way a change is implemented during development or maintenance, development has a much higher rate of change activity: these systems average over 1,000 changes during testing. Although the high number of changes may increase certain costs (e.g., configuration control), it may actually reduce others (e.g., testing is not repeated once for every change). Maintainers are not only generally unfamiliar with the systems they maintain, but the volume of maintenance may be insufficient to develop such familiarity. We expected the unfamiliarity with the maintained systems to have a more dramatic impact on the isolation activity (which might require an understanding of the entire system) than the completion activity (which typically requires only an understanding of individual modules). Instead, we discovered a proportional increase in both isolation and completion efforts (Figure 6). This may be explained by the fact that SEL maintainers are experts in the application domain, not software development; therefore, they may be expected to readily understand the change specifications, but not the code.

Both during development and maintenance a significant fraction of the changes are corrections (Figure 3). Figure 8 shows that the types of faults corrected during development and maintenance are similarly distributed (Figure 5, question 7). During maintenance, more corrections are related to incorrect initialization (21 vs. 17%) and logic (25 vs. 19%), but fewer are related to incorrect interface (19 vs. 22%), data (26 vs. 28%), and computation (9 vs. 14%) as compared to develop-

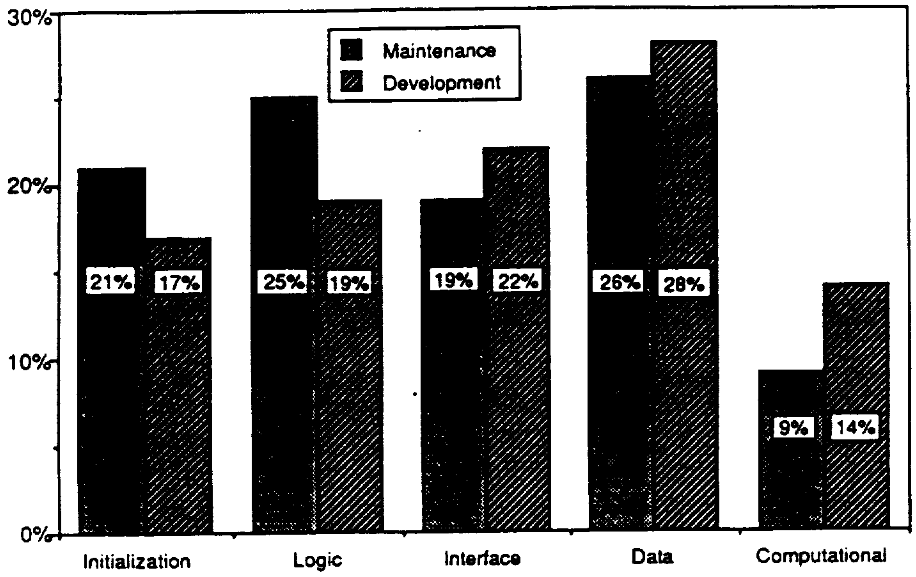


Figure 8. Number of faults per class: maintenance vs. development.

ment (see [20] for definition of classification scheme). Some of the differences seem to be related to the organizational structure of the environment. Maintenance is performed by people more familiar with the application domain and less familiar with the solution domain. The opposite is true for the developers. In this environment, many application-specific parameters are reflected in the software as initialization parameters. As such, they require a clear understanding of the application, and faults are more easily found by maintainers. The opposite is true for typical solution faults such as interface and computational faults.

Figure 9 shows that the distributions of errors differ significantly between maintenance and development (Figure 5, question 8). During maintenance, many more faults are attributed to inappropriate requirements or specifications (26 vs. 3%), and a few more are attributed to inappropriate design (11 vs. 8%): fewer are attributed to inappropriate implementation (55 vs. 79%) or previous changes (2 vs. 10%). In attempting to explain these differences, the following hypotheses have been formulated. Few faults are attributed to previous changes during maintenance because maintainers are unaware of changes made during development and the

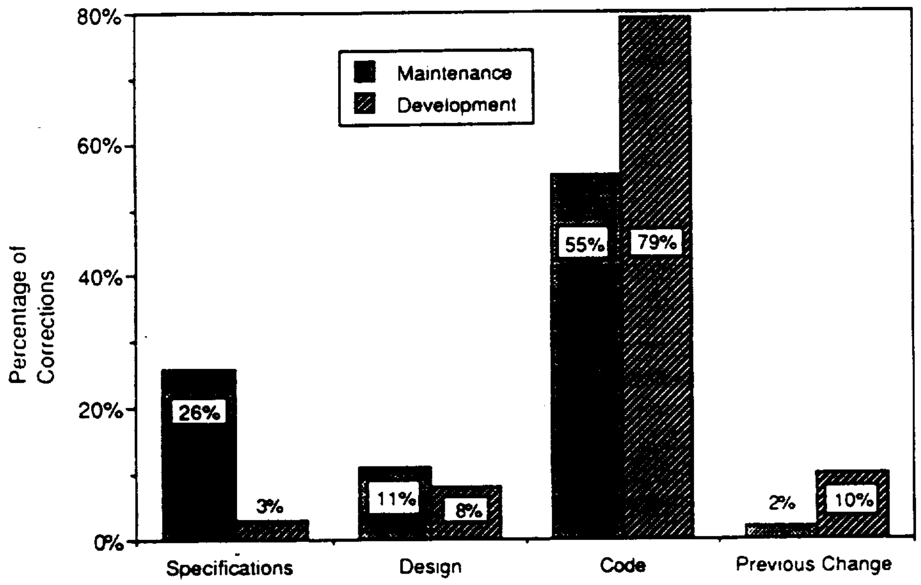


Figure 89. Number of faults per source: maintenance vs. development.

total number of changes during this phase is low. The high proportion of faults attributed to the requirements or specifications reflects the nature of the testing: applications experts are now using the systems to prepare for the missions, whereas during development most testing is performed by the developers themselves.

During development and maintenance, effort data is collected according to the following process model: isolation (understanding a requested change and identifying the affected modules), design (proposing a change), implementation (implement the proposed change), unit and system test (testing the changed modules and system), and acceptance test (testing a set of related changes). The development data include all effort; it is not limited to changes.

Figure 10 shows that during maintenance, more effort is spent on design activities, about the same amount of effort is spent on implementation activities, and less effort is spent on testing activities (Figure 5, question 9). The increase in design effort may be explained by a lack of familiarity with the system structure, resulting in increased effort to isolate changes. The decrease in testing effort may be explained by different testing procedures. During maintenance, integration testing is almost absent because the system structure doesn't change much, and acceptance testing is performed for groups of changes together.

How do these results compare with similar findings published in the literature? While comparing baseline data across environments is difficult, some patterns are evident. The increased cost of maintenance changes and corrections has been noted previously by many authors [22, 27]. This lends support to the claim that faults introduced during design but discovered during maintenance may cost significantly more than if discovered and corrected earlier in the life cycle [27]. As has been noted in other environments [28], we find that maintenance changes in the SEL require more "up-stream" (i.e., design) than "down-stream" (i.e., testing) effort).

4.3 Development for Maintenance

As a final result of the maintenance measurement program, the SEL has enhanced its understanding of the impact of development decisions on maintenance (Figure 11). This increased understanding is illustrated by our initial findings concerning the complexity of delivered products and the quality of their documentation. The qualitative results of this section are based primarily on subjective data from exploratory interviews. Nevertheless, they are essential during the early phases of a measurement program for guiding future improvement cycles.

Our initial inquiries have revealed complexity problems related to intermodule structure and the encoding global information (Figure 11, question 10). Maintainers reported major problems related to the fact that global information was encoded redundantly. For example, constants were encoded in multiple FORTRAN common blocks. Software modification frequently resulted in inconsistent representations of global information.

Two recurrent documentation problems have been identified (Figure 11, question 11). These concern the

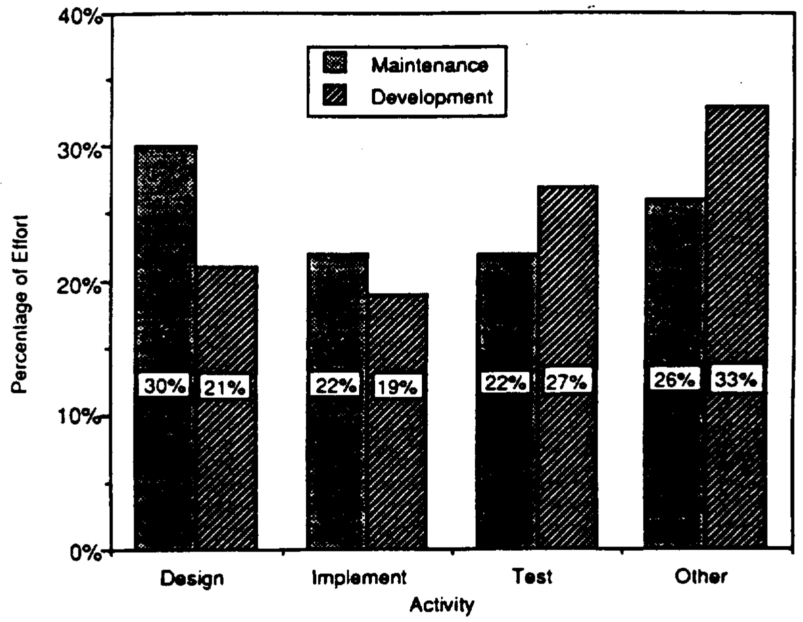


Figure 10. Effort per activity: maintenance vs. development.

GOAL 7: Characterize the impact of the delivered product on maintenance.
QUESTION 10
What structural product characteristics have positive/negative effects on maintenance?
QUESTION 11
What product documentation standards have positive/negative effects on maintenance?

Figure 11. Measurement goal for understanding the effects of development on maintenance.

use of program design language (PDL) and debug statements. PDL descriptions of each module are included in the source code as a header. Most maintainers regard PDL as redundant. Furthermore, the delivered PDL is usually outdated. In the SEL environment, developers are required to keep their design PDL as part of the software module. Unfortunately, this PDL is frequently obsolete by the time the module reaches the maintenance phase; thus, it is useless to the maintainers. Also, the majority of people maintaining the software suggested that this practice be stopped entirely, since the same level of abstraction is provided to them in the code structure and comments.

Many maintainers suggested that the debug interface of the code be improved. Because attitude ground support software is highly computational, an extensive debug interface is provided with each system. The problem with the current debug interface is that frequently it assumes intimate familiarity with the code in that the output was of the form ⟨**variable**⟩ = ⟨**value**⟩. Maintainers suggested that future debug interfaces provide a more descriptive explanation of the output printed.

As we learn more about the problems maintainers have with the software delivered from development and identify solutions to these problems, the guidelines and standards for development [7-9] will be modified to reflect these recommendations.

5. SUMMARY AND CONCLUSIONS

In this section, we summarize the benefits of the maintenance measurement study for the SEL, outline future maintenance measurement directions within SEL, and package some of the general lessons learned about establishing measurement programs for use in other maintenance environments.

5.1 SEL Maintenance Study Benefits

The most immediate benefit of this program has been an enhanced understanding of the SEL maintenance environment. The quantitative baselines presented in the preceding section resulted in a better understanding of maintenance requests, maintained products, and maintenance processes. They enabled us to identify weaknesses in the SEL maintenance environment.

The comparison between changes performed during development and maintenance has helped us understand where we may benefit from existing development baselines. For example, whereas the distributions of faults corrected during development and maintenance are similar, effort distributions are not. This suggests that reuse of lessons learned from development is more justified when they pertain to faults than when they pertain to effort.

Baselines may also be used to compare the effects of new development technologies on maintenance. For example, both cleanroom and an Ada/object-oriented design approach have been applied on recent development projects with the expectation that "more reliable" systems will result. We are now in a position to validate these expectations by comparing the effects of the new approaches to traditionally run projects.

In the long term, development and maintenance are expected to improve as a result of our increased understanding. At this point, recommendations for improvement are based predominantly on qualitative feedback from maintainers (rather than quantitative measurement baselines). Most of these suggestions have to do with the separation of the development and analysis organizations (Figure 1) and the absence of standard maintenance processes. The separation of development and maintenance means that a maintainer is entirely dependent on the code and documentation acquired at the time of delivery [29]. Consequently, inadequacies in the code or documentation are much more of an obstacle to maintenance than in an organization where maintenance and development are more closely related. Each maintenance change is performed by one individual without much guidance regarding the maintenance process itself. The ad hoc nature of the maintenance processes makes it hard to measure, compare measurements, and make recommendations. We expect our measurement program to contribute to the standardization of maintenance processes over time.

Overall, the SEL maintenance measurement program is perceived as successful and beneficial to this particular environment. The lessons learned from our study have resulted in changes and additions to the SEL standards and policies for software development [8]. Because numerous new projects are always under development in the SEL, we will be able to examine whether the revised standards have a measurable impact on the quality of the development product.

5.2 Future Maintenance Research

As we continue to learn about the SEL maintenance environment, numerous future measurement directions become evident. Some directions reflect changes in the environment itself, others reflect changes in our understanding of the environment. We must continually revise our goals, questions, metrics, and procedures to reflect the current priorities and understanding. Figure 12 contains an example set of revised questions for each of our seven maintenance goals to guide future maintenance studies.

We must continue to revise our measurement program in response to previous misconceptions inherent in our initial qualitative models of maintenance process. For example, our current effort classification scheme does not explicitly recognize configuration management as a discrete activity. This effort is grouped together with nontechnical activities such as meetings and management. In the future, we may want to update our data collection forms to include configuration management as a separate activity, since it seems to represent a significant portion of current maintenance effort.

GOAL 1: Characterize the changes performed during maintenance.
QUESTION 1
How many changes of each type are requested by different sources (e.g., analyst, operator)?
GOAL 2: Characterize product evolution during maintenance.
QUESTION 2
How does coupling/cohesion change during maintenance?
GOAL 3: Characterize the maintenance process stability.
QUESTION 3
Which process factors determined process stability (e.g., staffing level, familiarity with system)?
GOAL 4: Compare changes made during development and maintenance.
QUESTION 4
What is the average change effort per module during each phase?
GOAL 5: Compare changes made to products at both phases.
QUESTION 5
What are the distributions of requirements changes by type?
GOAL 6: Compare development and maintenance processes.
QUESTION 6
What are the distributions of change effort by activity.
GOAL 7: Characterize the impact of the delivered product on maintenance.
QUESTION 7
What product characteristics resulting from reuse have positive/negative effects on maintenance?

Figure 12. Revised measurement questions for future maintenance improvement cycles.

When our empirical investigations identify important phenomena, we must refocus our measurement goals and questions in order to study the phenomena. For example, one hypothesized implication of the stable architecture of the maintained systems (very few modules are being added or deleted) is that module cohesion within these systems may be deteriorating. Such deterioration may lead to weaker and weaker system architecture, and ultimately lead to even more difficult maintenance. Such a hypothesis needs much closer investigation before it can be presented as a potential problem.

When measurement does identify specific problems, the next step is to analyze the problems and attempt to identify viable solutions. For instance, we have quantified the types and kinds of faults uncovered during maintenance. Next, we might begin to analyze their causes in development. Such analysis may lead us to mechanisms for preventing faults, or it may help us identify better ways of detecting them.

Finally, the maintenance environment itself is continually changing. Transitions to the use of Ada and Cleanroom development in the SEL will require periodic adjustments to our measurement procedures. Such changes are not unexpected; in fact, measurement by nature must continue to evolve as the environment evolves.

5.3 Measurement Lessons Learned

The extension of the SEL into maintenance not only enabled us to gain experience with maintenance but also with establishing a maintenance measurement program [25].

Our first lesson is that there is a distinction, at least conceptually, between start-up and routine phases of measurement. During the start-up phase, there is considerable freedom to reevaluate measurement goals and redesign the metrics and procedures as our understanding of the local priorities and what is feasible grows. Once data collection forms have been designed and reflected in the data base and once people have been instructed in the procedures, it becomes expensive to introduce further changes. It is therefore critical that the start-up phase proceed cautiously. We suggest validating all measurement procedures through pilot studies.

Our second lesson concerns which questions are suitable for routine measurement. It may be tempting to use routine measurement as a mechanism for answering questions that could be resolved more efficiently by other means. For example, if the software design documentation is never maintained, it would be wasteful to discover this via routine data collection. Routine measurement is appropriate for monitoring large-scale and historical trends, but it is not needed to ascertain simple facts. Many of the questions we would like to pursue are risky, i.e., we cannot be sure that the resulting data will prove useful.

Third, we have found the establishment of a measurement program in a new environment to be a time-consuming and sensitive task. Getting the program started requires building initial models of the maintenance organization, the maintained products, the maintenance processes, and the specific maintenance problems at hand. These models are used to design the measurement procedures, but must be validated during the start-up phase. Special care must also be taken to establish the creditability of measurement and win the cooperation needed to make the program a success. To collect valid data, the people providing most of the data need to be well motivated and instructed. Motivation requires addressing measurement goals of direct interest to the people providing cooperation and an opportunity for these people to review and comment on the resulting data and analyses.

Our analysis results demonstrate the immediate returns possible from investment in a measurement program. A measurement program provides invaluable insight into the processes and products within the given environment. As long as measurement is performed within a context of well defined goals and questions, such a program can be a success for any software organization.

ACKNOWLEDGMENTS

Research for this study was supported by National Aeronautics and Space Administration grant NSG-5123 to the University of Maryland.

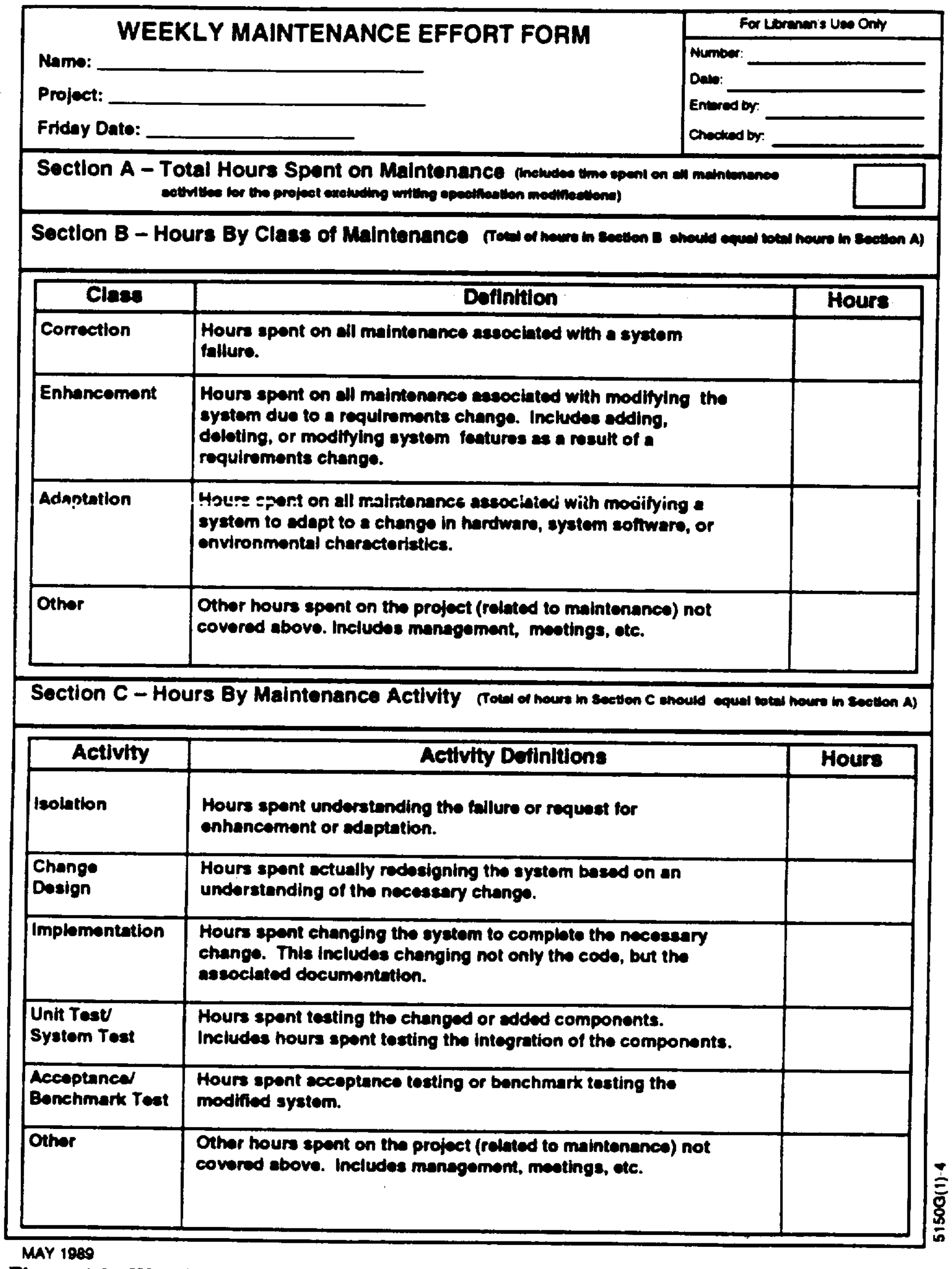
We thank the CSC and GSFC personnel who have participated in our maintenance measurement program—the project personnel, the SEL data librarians, and those who reviewed earlier versions of this paper—for their tremendous support. We also thank Bruce Blum, the editor in charge of our paper, and the anonymous referees for their excellent suggestions.

REFERENCES

1. V. R. Basili, Software development: a paradigm for the future, in *Proceedings of the 13th Annual International Computer Software and Applications Conference*, 1989, pp. 471-485.
2. M. Buhler and J. Valett, Annotated Bibliography of Software Engineering Laboratory Literature, SEL-82-906, NASA/GSFC, Greenbelt, Maryland, 1990.
3. V. R. Basili, Measuring the software process and product: lessons learned in the SEL, in *Proceedings of the 10th Annual Software Engineering Workshop*, 1985.
4. F. E. McGarry, Studies and experiments in the SEL, in *Proceedings of the 10th Annual Software Engineering Workshop*, 1985.
5. R. W. Selby, Jr., and V. R. Basili, Comparing the Effectiveness of Software Testing Strategies, *IEEE Trans. Software Eng.* 13:1278-1296 (1987).
6. J. D. Valett and F. E. McGarry, A summary of software measurement experience in the Software Engineering Laboratory, in *Proceedings of the 21st Annual Hawaii International Conference on System Sciences*, 1988.
7. F. McGarry, G. Page, S. Eslinger, V. Church, and P. Merwarth, Recommended Approach to Software Development, SEL-81-205, NASA/GSFC, Greenbelt, Maryland, 1983.
8. Manager's Handbook for Software Development, rev. 1, SEL-84-101, NASA/GSFC, Greenbelt, Maryland, 1990.
9. R. Wood and E. Edwards, Programmer's Handbook for Flight Dynamics Software Development, SEL-86-001, NASA/GSFC, Greenbelt Maryland, 1986.
10. V. R. Basili and D. M. Weiss, A Methodology for Collecting Valid Software Engineering Data, *IEEE Trans. Software Eng.* SE-10, 728-738 (1984)
11. V. R. Basili and H. D. Rombach, The TAME Project: Towards Improvement-Oriented Software Environments, *IEEE Trans. Software Eng.* 758-773 (1988).
12. R. B. Grady, Measuring and Managing Software Maintenance, *IEEE Software* 4, 35-45 (1987).
13. R. Arnold and D. Parker, The dimensions of healthy maintenance, in *6th International Conference on Software Engineering*, 1982, pp. 10-27.
14. L. Belady and M. Lehman, A Model of Large Program Development, *IBM Syst. J.* 3, 225-252 (1976).
15. H. D. Rombach, A Controlled Experiment on the Impact of Software Structure on Maintainability, *IEEE Trans. Software Eng.* SE-12, 344-354 (1987).
16. C. K. S. Chong Hok Yuen, An empirical approach to the study of errors in large software under maintenance, in *Conference on Software Maintenance – 1985*, 1985, pp. 96-105.
17. H. D. Rombach and V. R. Basili, Quantitative assessment of maintenance: an industrial case study, in *IEEE Conference on Software Maintenance – 1987*, 1987, pp. 134-144.
18. B. W. Boehm and P. N. Papaccio, Understanding and Controlling Software Costs, *IEEE Trans. Software Eng.* SE-14, 1462-1477 (1988).
19. D. P. Hale and D. A. Haworth, Software maintenance: a profile of past empirical research, in *Conference on Software Maintenance – 1988*, 1988, pp. 236-240.
20. G. Heller, Data Collection Procedures for the Rehosted SEL Database, SEL-87-008, NASA/GSFC, Greenbelt, Maryland, 1987.
21. M. So, SEL Database Organization and User's Guide, SEL-89-001, NASA/GSFC, Greenbelt, Maryland, 1989.
22. H. D. Mills, Software Development, *IEEE Trans. Software Eng.* 2:265-273 (1976).
23. C. Brophy, Lessons Learned in the Transition to Ada from FORTRAN at NASA/Goddard, SEL-89-005, NASA/GSFC, Greenbelt, Maryland, 1989.

24. S. Green, A. Kouchakdjian, V. Basili, and D. Weidow, The Cleanroom Case Study in the Software Engineering Laboratory: Project Description and Early Analysis, SEL-90-002, NASA/GSFC, Greenbelt, Maryland, 1990.
25. H. D. Rombach and B. T. Ulery, Establishing a measurement based maintenance improvement program: lessons learned in the SEL, *Conference on Software Maintenance – 1989*, 1989, pp. 50-57.
26. E. B. Swanson, The dimensions of software maintenance, in *Proceedings of the 2nd IEEE International Conference on Software Engineering*, 1976, pp. 492-497.
27. B. Boehm, *Software Engineering Economics*, Prentice-Hall, Englewood Cliffs, New Jersey, 1981.
28. N. Chapin, Software Maintenance Life Cycle, in *Conference of Software Maintenance – 1988*, 1988, pp. 6-13.
29. E. B. Swanson and C. M. Beath, Departmentalization in Software Development and Maintenance, *Commun. ACM* 33, 658-667 (1990).

APPENDIX A: Data Collection Forms



WEEKLY MAINTENANCE EFFORT FORM

Name: ____________________

Project: ____________________

Friday Date: ____________________

For Librarian's Use Only

Number: ____________

Date: ____________

Entered by: ____________

Checked by: ____________

Section A – Total Hours Spent on Maintenance (includes time spent on all maintenance activities for the project excluding writing specification modifications)

Section B – Hours By Class of Maintenance (Total of hours in Section B should equal total hours in Section A)

Class	Definition	Hours
Correction	Hours spent on all maintenance associated with a system failure.	
Enhancement	Hours spent on all maintenance associated with modifying the system due to a requirements change. Includes adding, deleting, or modifying system features as a result of a requirements change.	
Adaptation	Hours spent on all maintenance associated with modifying a system to adapt to a change in hardware, system software, or environmental characteristics.	
Other	Other hours spent on the project (related to maintenance) not covered above. Includes management, meetings, etc.	

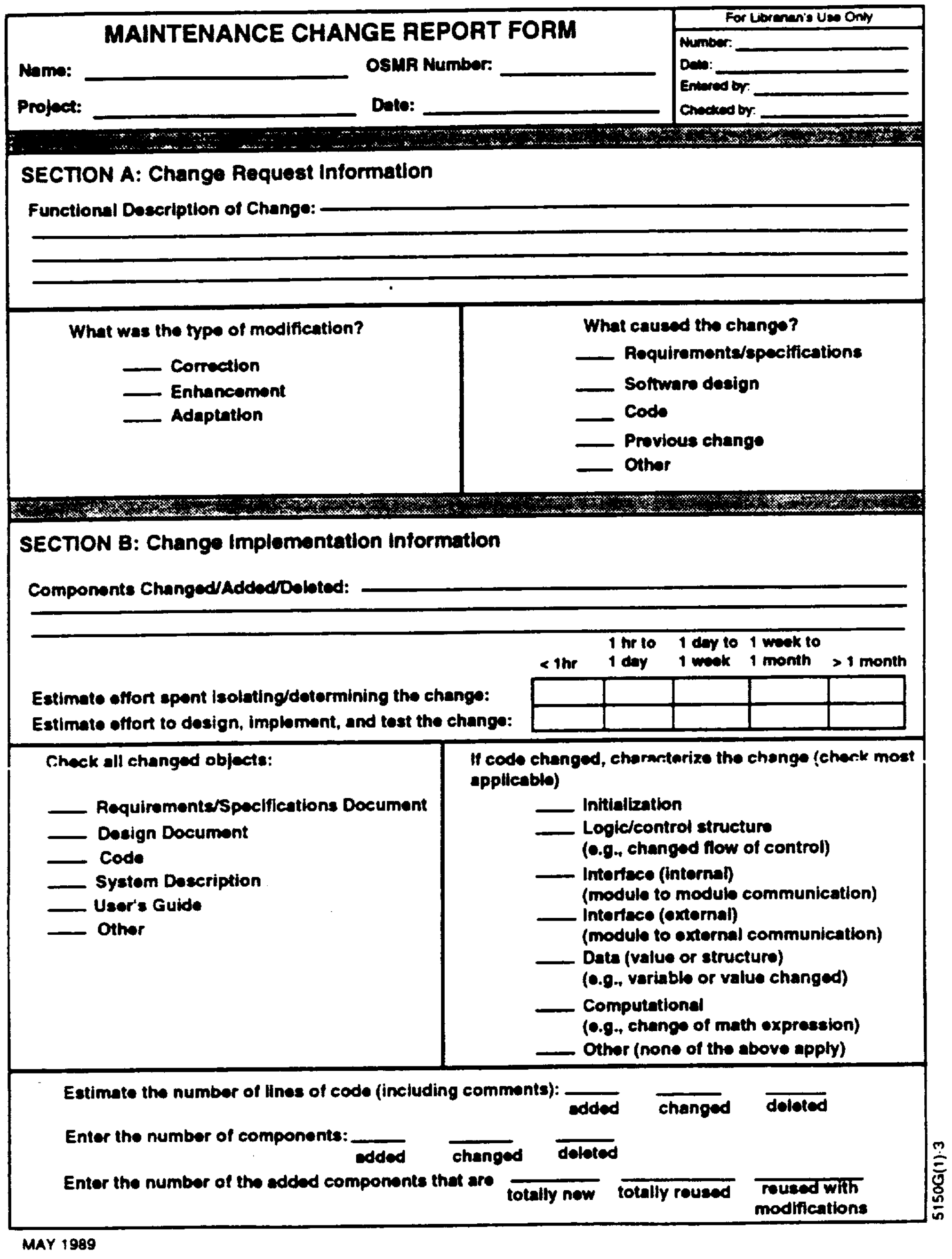
Section C – Hours By Maintenance Activity (Total of hours in Section C should equal total hours in Section A)

Activity	Activity Definitions	Hours
Isolation	Hours spent understanding the failure or request for enhancement or adaptation.	
Change Design	Hours spent actually redesigning the system based on an understanding of the necessary change.	
Implementation	Hours spent changing the system to complete the necessary change. This includes changing not only the code, but the associated documentation.	
Unit Test/ System Test	Hours spent testing the changed or added components. Includes hours spent testing the integration of the components.	
Acceptance/ Benchmark Test	Hours spent acceptance testing or benchmark testing the modified system.	
Other	Other hours spent on the project (related to maintenance) not covered above. Includes management, meetings, etc.	

5150G(1)-4

MAY 1989

Figure A1. Weekly Maintenance Report Forms.



MAINTENANCE CHANGE REPORT FORM

Name: ____________ OSMR Number: ____________

Project: ____________ Date: ____________

For Librarian's Use Only
Number: ____________
Date: ____________
Entered by: ____________
Checked by: ____________

SECTION A: Change Request Information

Functional Description of Change: ____________

What was the type of modification?

- ___ Correction
- ___ Enhancement
- ___ Adaptation

What caused the change?

- ___ Requirements/specifications
- ___ Software design
- ___ Code
- ___ Previous change
- ___ Other

SECTION B: Change Implementation Information

Components Changed/Added/Deleted: ____________

	< 1hr	1 hr to 1 day	1 day to 1 week	1 week to 1 month	> 1 month
Estimate effort spent isolating/determining the change:					
Estimate effort to design, implement, and test the change:					

Check all changed objects:

- ___ Requirements/Specifications Document
- ___ Design Document
- ___ Code
- ___ System Description
- ___ User's Guide
- ___ Other

If code changed, characterize the change (check most applicable)

- ___ Initialization
- ___ Logic/control structure (e.g., changed flow of control)
- ___ Interface (internal) (module to module communication)
- ___ Interface (external) (module to external communication)
- ___ Data (value or structure) (e.g., variable or value changed)
- ___ Computational (e.g., change of math expression)
- ___ Other (none of the above apply)

Estimate the number of lines of code (including comments): ___ added ___ changed ___ deleted

Enter the number of components: ___ added ___ changed ___ deleted

Enter the number of the added components that are ___ totally new ___ totally reused ___ reused with modifications

5150G(1)-3

MAY 1989

Figure A2. Maintenance Change Report Form.