Technical Report, University of Maryland, Computer Science Dept., College Park, MD 20770 USA. Aug. 1995.

Understanding and Predicting the Process of Software Maintenance Releases*

Victor Basili, Lionel Briand, Steven Condon,
Yong-Mi Kim, Walcélio L. Melo and Jon D. Valett**

To appear in the
***Proc. of the 18th IntÕl Conf. on S/W Eng.*, Berlin, Germany, March 1996.**

Abstract

One of the major concerns of any maintenance organization is to understand and estimate the cost of maintenance releases of software systems. Planning the next release so as to maximize the increase in functionality and the improvement in quality are vital to successful maintenance management. The objective of this paper is to present the results of a case study in which an incremental approach was used to better understand the effort distribution of releases and build a predictive effort model for software maintenance releases. This study was conducted in the Flight Dynamics Division (FDD) of NASA Goddard Space Flight Center (GSFC). This organization is representative of many other software maintenance organizations. Over one hundred software systems totaling about 4.5 million lines of code, are maintained by this organization. Many of these systems are maintained for many years and regularly produce new releases. This paper presents three main results: 1) a predictive effort model developed for the FDD's software maintenance release process, 2) measurement-based lessons learned about the maintenance process in the FDD, 3) a set of lessons learned about the establishment of a measurement-based software maintenance improvement program. In addition, this study provides insights and guidelines to obtain similar results in other maintenance organizations.

Key words: software maintenance, measurement, experience factory, case studies, quality improvement and goal/question/metric paradigms.

1. Introduction

1.1 Issues

Software maintenance is generally recognized to consume the majority of resources in many software organizations [Abran & Nguyenkim, 1991; Harrison & Cook, 1990]. As a result, planning releases so as to maximize functionality and quality within the boundaries of resource constraints (such as, budget, personnel, and time to market) is vital to the success of an organization. The software maintenance process is, however, still poorly understood and loosely managed worldwide. As described in [Haziza *et al.*, 1992], numerous factors can affect software

---

* V. Basili, Y.-M. Kim and W. Melo are with the University of Maryland, Institute for Advanced Computer Studies and Computer Science Dept., A. V. Williams Bldg., College Park, MD 20742 USA. S. Condon is with Computer Sciences Corporation, 10110 Aerospace Rd., Lanham-Seabrook, MD 20706 USA. L. Briand is with the CRIM, Montreal, Canada. J. Valett is with NASA Goddard Space Flight Center, Software Engineering Branch, Greenbelt, MD 20771 USA. E-mails: {basili | kimy | melo}@cs.umd.edu., lbriand@crim.ca, steven_condon@cscmail.csc.com, jon.valett@gsfc.nasa.gov

** Authors are listed in alphabetical order.

maintenance quality and productivity, such as process, organization, experience, and training. Unfortunately the complexity of the phenomena frequently obscures the identity and impact of such factors in any given maintenance organization. The resulting uncertainty about productivity and quality in the next software release gives rise to unreliable cost and schedule release estimates.

To effectively manage the software release process, managers must be supplied with more accurate information and more useful guidelines to aid them in improving the decision-making process, planning and scheduling maintenance activities, foreseeing bottlenecks, allocating resources, optimizing the implementation of change requests by releases, etc. In order to accomplish this, we need to define and validate methodologies that take into account the specific characteristics of a software maintenance organization and its processes, e.g., the software maintenance release process. However, methods that help software maintainers change large software systems on schedule and within budget are scarce. Methods currently available for improving software processes, such as the Software Engineering Institute Capability Maturity Model (SEI CMM) [Paulk, *et al*., 1993], have not been validated thoroughly. Even though a few methods have been demonstrated to be useful for software development (e.g., QIP [Basili & Rombach, 1988]) they have only recently begun to be applied to software maintenance [Valett *et al.*, 1994]. The work described in this paper is a further step in the application of these methods.

1.2 Objective

The objective of this paper is to use an incremental and inductive approach for improving software maintenance by focusing on the construction of descriptive and predictive models for software maintenance releases. We present the results of a case study in which this approach was successfully used to build a predictive effort model for software maintenance releases in a large-scale software maintenance organization. This case study took place in the Flight Dynamics Division (FDD) of the NASA Goddard Space Flight Center (GSFC). This organization is a representative sample of many other software maintenance organizations. The FDD maintains over one hundred software systems totaling about 4.5 million lines of code, and many of these systems are maintained for many years and regularly produce new releases.

In this paper, we are mostly concerned with presenting the results of the process used to build descriptive and predictive models of software maintenance releases in a particular environment. Although, the models produced in this study are organization dependent, we believe that the process used to build them can be easily replicated in different software organizations.

The paper is organized as follows. It first presents the framework in which this study was conducted: the FDD and the Software Engineering Laboratory (SEL). Next an overview of our approach to software maintenance process improvement is provided. The paper then presents the measurement program used to collect product and process data about maintenance projects and releases. This is followed by a quantitative analysis of the data collected from January 1994 to June 1995 on the delivery process of over 29 releases of 11 different systems. This analysis presents descriptive models of the maintenance environment, as well as a predictive model for release productivity. Next the paper presents the lessons learned from the analysis and validation of data, and discusses lessons drawn from establishing a software maintenance measurement program. Finally, future work is outlined.

2. Study framework and approach to model building

2.1 The environment

GSFC manages and controls NASA's Earth-orbiting scientific satellites and also supports Space Shuttle flights. For fulfilling both these complex missions, the FDD developed and now maintains over 100 different software systems, ranging in size from 10 thousand source lines of code (KSLOC) to 250 KSLOC, and totaling approximately 4.5 million SLOC. Many of these systems are maintained over many years and regularly produce new releases. Of these systems, 85% are written in FORTRAN, 10% in Ada, and 5% in other languages. Most of the systems run on IBM mainframe computers, but 10% run on PCs or UNIX workstations.

This study was conducted through the SEL, which is a joint-venture between GSFC, Computer Sciences Corporation, and the University of Maryland. Since 1976, the SEL has been modeling and experimenting in the FDD with the goals of understanding the software development process in this environment; measuring the effect of software engineering methodologies, tools, and models on this process; and identifying and applying successful practices [McGarry *et al.*, 1994]. Recently, responding to an organizational need to better control the cost and quality of software maintenance, the SEL has initiated a program aimed at characterizing, evaluating and improving these maintenance processes.

2.2 The approach

This SEL program on maintenance began in October 1993 and is being conducted using an empirical approach which is an instantiation of the more general Quality Improvement Paradigm (QIP) and the Goal/Question/Metric Paradigm (GQM) [Basili & Rombach, 1988]. In the following paragraphs we provide an overview of this approach and show how it has helped us in the construction of a predictive model for software maintenance releases. This approach was tested and continuously refined through experience. Further details can be found in [Briand *et al*, 1994; 1995].

First, qualitative studies were performed in order to better comprehend organization- and process-related issues. Here, the objective was to identify and understand, as objectively as possible, the real issues faced by the organization. Specific modeling techniques such as the Agent Dependency Model were used as part of this step (see [Briand *et al.*, 1995]). Such a technique can help capture important properties of the organizational context of the maintenance process and help to understand the cause-effect mechanisms leading to problems. Such qualitative data must be complemented with quantitative data.

In a subsequent step, the outputs produced by the first step were used to justify and define a relevant and efficient measurement program (i.e., what to collect, when to collect, and how to collect). In addition, interpreting the data coming from such a program was made easier because of the increased level of understanding of the process in place.

Once the measurement program began (i.e., data collection forms were available, data collection procedures defined, people trained, etc.), process and product data were collected and various issues identified as relevant to the maintenance process were analyzed. Based upon such analyses, the relationships between process attributes, such as effort, and other variables characterizing the changes, the product to be changed, and the change process were identified. For instance, in this paper, a model for predicting release effort from estimated release size is presented to help software maintenance managers in the FDD environment optimize release resource expenditures. Such models will be incrementally refined when new information of either a qualitative or quantitative nature is available.

3. A GQM for this study

As pointed out in [Pigoski, & Nelson, 1994; Rombach *et al.*, 1992; Schneidewind, 1994], the establishment of a measurement program integrated into the maintenance process, when well defined and established, can help us acquire an in-depth understanding of specific maintenance issues and thereby lay a solid foundation for the improvement of the software maintenance release processes. To do so, we must define and collect those measures that would most meaningfully characterize the maintenance process and products. In order to define the metrics to be collected during the study, we used the GQM paradigm [Basili & Rombach, 1988]. We first present the GQM goals of the study, then present the metrics and the data collection method used. For the sake of brevity the questions accompanying each goal are presented with the data analysis.

3.1 Goals

Goal 1: Analyze: the maintenance release generation process
for the purpose of: characterization
with respect to: effort
from the point of view of: management, experience factory for maintenance

In this goal, we are interested in understanding the maintenance release generation process of the maintenance organization with respect to the distribution of effort across software activities, across maintenance change types, and across software projects.

Next , we need to identify the variables we can use to produce predictive models for maintenance. That is, we must study and understand the relationship between the different facets of effort and other metrics, such as type of releases, type of change, change size, types of component change (modification, inclusion or deletion of code). Formalizing such a problem in the GQM format, we formulate the following goal:

Goal 2: Analyze: maintenance release process
for the purpose of: identifying relationships between effort and other variables
with respect to: type of release, type of change, size of change, and kind of change
from the point of view of: experience factory for maintenance

This second goal is a necessary step that leads from Goal 1 to the following goal:

Goal 3: Analyze: release delivery process
for the purpose of: prediction
with respect to: productivity
from the point of view of: experience factory for maintenance

3.2 Metrics and models

In this section we describe the metrics and models used in this study. The preliminary qualitative modeling of the maintenance process enabled the definition and refinement of these metrics and models.

Maintenance change types

We consider the following maintenance change types:

- error correction: correct faults in delivered system.
- enhancement: improve performance or other system attributes, or add new functionality.
- adaptation: adapt system to a new environment, such as a new operating system.

Maintenance activities

The following maintenance activity classification is used in the data collection forms:

- Impact analysis/cost benefit analysis. The number of hours spent analyzing several alternative implementations and/or comparing their impact on schedule, cost, and ease of operation.
- Isolation. The number of hours spent understanding the failure or request for enhancement or adaptation.
- Change design. The number of hours spent actually redesigning the system based on an understanding of the necessary change; includes semiformal documentation, such as release design review documents.
- Code/unit test. The number of hours spent to code the necessary change and test the unit; includes semiformal documentation, such as software modification test plan.
- Inspection/certification/consulting. The number of hours spent inspecting, certifying, and consulting on another's design, code, etc., including inspection meetings.
- Integration test. The number of hours spent testing the integration of the components.
- Acceptance test. The number of hours spent acceptance testing the modified system.
- Regression test. The number of hours spent regression testing the modified system.
- System documentation. The number of hours spent writing or revising the system description document and math specification.
- User/other documentation. The number of hours spent writing or revising the user's guide and other formal documentation, except system documentation.
- Other. The number of hours spent on activities other than the ones above, including management.

A more detailed presentation of the maintenance activities model is presented in [Valett *et al*, 1994].

Release types

Maintenance releases in our environment were classified into three categories: mostly error correction, mostly enhancement, and mixture. A more detailed discussion is presented in Section 4.4

Size and effort

The size of a software change is measured as the sum of the number of source lines of code (SLOC) added, changed, and deleted. SLOC is defined to include all code, unit header lines, comments, and blank lines. Effort is measured by person hours that were charged to maintenance projects.

3.3 Data collection method

The following forms were used to collect the data for this study:

- software change request (SCR) form;
- weekly maintenance effort form (WMEF);
- software release estimate form (SREF).

Again, without a preliminary qualitative analysis of the maintenance process, determining the content and format of the forms would have been extremely difficult.

3.3.1 SCR forms

On the SCR, the user or tester specifies what *type of change* is being requested: *error correction*, *enhancement*, or *adaptation*. The maintainer specifies using an ordinal scale the effort spent isolating/determining the change, as well as the effort spent designing/implementing/testing the change. The maintainer also provides six numbers characterizing the extent of the change made: (1) number of SLOC added, (2) changed, (3) deleted; (4) number of components added, (5) changed, (6) deleted. In addition, the maintainer further specifies how many of the components in item (4) were newly written, how many were borrowed and reused verbatim, and how many were borrowed and reused with modification

3.3.2 WMEF forms

Each maintainer, tester, and manager working on one of the study projects was required to report project hours each week on a WMEF. The WMEF required each person to break down project effort two ways: (1) by specifying the hours by the type of change request performed (error corrections, enhancements, or adaptations) or as *other* hours (e.g., management, meetings), and (2) by specifying the hours by the software activities performed (such as design, implementation, acceptance testing).

Because the WMEF did not originally allow a person to specify to which maintenance release of the project his hours applied, uncertainty resulted if a maintenance team was involved in more than one maintenance release in the same week. For many projects, maintenance releases did overlap. Therefore, in August 1994, we revised the WMEF by requiring personnel in the study to specify to which release each activity hour applied. In addition, each maintainer (but not tester) is now required to specify on his WMEF to which SCR each activity hour applies.

3.3.3 SREF forms

The SREF is a new form created by the authors to capture estimates of the release schedule, release effort, release content (i.e., list of SCRs), and release extent (i.e., number of units and lines of code to be added, changed or deleted). Maintenance task leaders submit an SREF at the end of each phase in the maintenance release life cycle

4. Quantitative analysis of the SEL sample maintenance goals

In this section, we provide the results of our analyses from the data collected during this study. In most cases, the data consisted of 25 complete releases for ten different projects. The effort per release ranged from 23 hours to 6701 hours, with a mean of 2201 hours. The total changes per release ranged from 21 SLOC to 23,816 SLOC, with a mean of 5654 SLOC.

4.1 Effort across maintenance activities

In this section we are interested in the following questions related to Goal 1:

- Q1.1. What is the distribution of effort across maintenance activities (i.e., analysis/isolation, design, implementation, testing, and other; see below)?
- Q1.2. What are the most costly projects and what is the distribution of effort across maintenance activities in these projects?

For simplicity, we have grouped the 12 maintenance categories into 5 groups, as follows:

- Analysis/isolation: impact analysis/cost benefit analysis, isolation
- Design: change design, 1/2 (inspection/certification/consulting)
- Implementation: code/unit test, 1/2 (inspection/certification/consulting)
- Testing: integration test, regression test, acceptance test
- Other: system documentation, other documentation, other

Using these groupings, the distribution of maintenance effort across maintenance activities is shown in Figure 1. The first pie chart of this figure represents the overall distribution based on the total effort expended in the 25 complete releases (10 projects) studied. Five projects accounted for 17 of these 25 releases. The remaining pie charts show the effort distributions for these 5 projects, based on their 17 complete releases. These 5 projects were the costliest projects in the FDD between January 1994 and June 1995, counting all project effort, i.e., including effort for both complete and partial releases in this time period . During this time period, Swingby accounted for 28% of the maintenance effort, MTASS for 19%, GTDS for 12%, MSASS for 10%, and ADG for 8%.

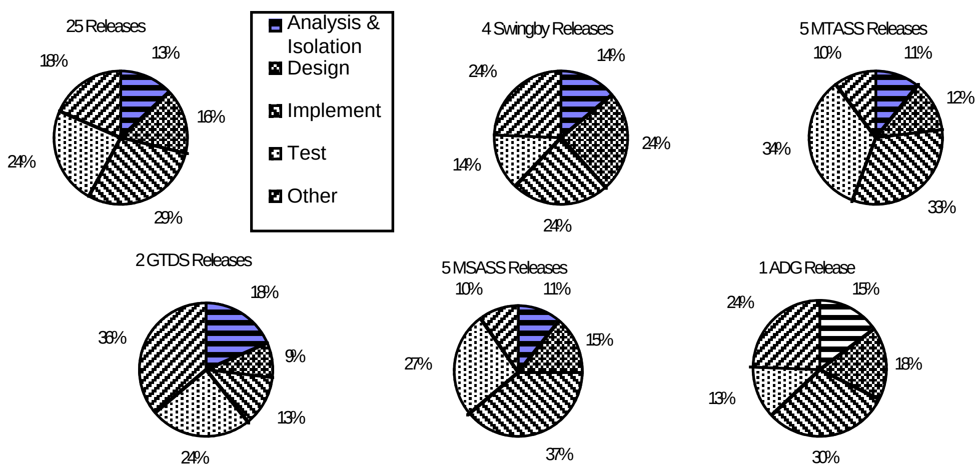

Figure 1: Distribution of effort among software maintenance activities

One difference amount these projects is that MTASS has the largest percentage of testing effort, 34%. Closer examination reveals that testing made up 17% of the two earlier MTASS releases and 43% of the three latter releases. A similar trend is suggested by MSASS—17%, followed by 31%. In addition, both projects show a decreasing trend in implementation effort, 43% followed by 27% for MTASS, and 41% followed by 37% for MSASS. These trends are not evident, however, for Swingby, the only other project in our study that is represented by more than 2 releases. The increase in MTASS and MSASS testing may be due to the fact that these programs consist of large software libraries that are enhanced and reused from mission to mission. As the software grows, more regression test time is necessary. Another difference is seen in the large amount of ‘other’ time for GTDS. One of the GTDS releases involved porting the GTDS software from an IBM mainframe to a workstation. A significant amount of training time (listed as other) may have been necessary for the maintainers. More study is required before we can with confidence recommend such pie charts to release managers as guides for resource allocation. It is likely that such models will also need to factor in what type of changes (adaptation, correction, or enhancement) constitute the release.

4.2 Effort across maintenance change types

In this section we consider the following questions related to Goal 1:

- Q1.3. What is the distribution of effort across maintenance change types (i.e., adaptation, error correction, enhancement, other)? That is, how was the total maintenance effort expended?
- Q1.4. Is the distribution of effort across maintenance activities the same for the different software maintenance change types?

Figure 2 presents the average distribution of effort across maintenance change types. The distributions for individual projects vary significantly from each other and also from this average distribution. For example, effort spent on enhancements varied from 51% to 89% among the most dominant projects.

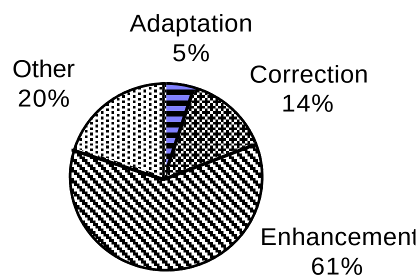

Figure 2: Effort Distribution by Type of Change

In the FDD, enhancements typically involve more SLOC than error corrections. The 25 complete releases contained 187 change requests from users. Of these, 84 were enhancement change requests, with a mean size of 1570 SLOC, whereas 94 were error correction change requests, with a mean size of only 61 SLOC. This data supports the intuitive notion that error corrections are relatively small isolated changes, while enhancements are larger changes to the functionality of the system.

Now, we address the fourth question. In order to answer this question, we need to know how a maintainer's activity effort is distributed for each change type. With the old WMEF we could not simultaneously analyze effort by both activity and change type. With the new WMEF we can do so for the programmers' effort, because programmers report the activity effort associated with each SCR, and we know the change type of each SCR. Due to the fact that testers, and usually task leaders, report their effort by release—but not by SCR—we cannot analyze their effort this way. Figure 3 shows the effort spent by programmers on correction and enhancement maintenance types, each broken down by maintenance activities. We do not include the 'testing' and 'other' groups of activities, because much of this activity is not tagged to individual SCRs, and we do not want to present a misleading picture of how much time is spent in these activities. As expected, software maintainers spent more effort on isolation activities when correcting code than when enhancing it. Conversely, they spent much more time on inspection, certification, and consulting, when enhancing code than when correcting it. The proportions of effort spent on design and code/unit test are almost the same for the two types of change requests.

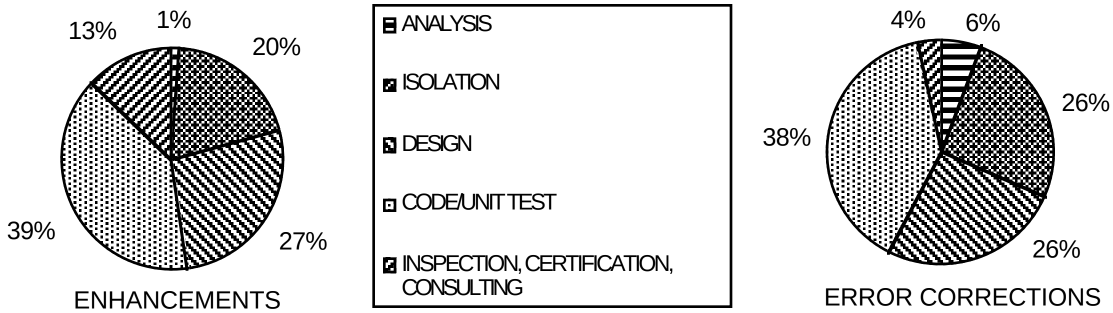

Figure 3. Programmer effort distribution across five maintenance activities for error correction and enhancement maintenance changes

4.3 Testing changes vs. release changes

In this section we consider the following question related to Goal 1:

- Q1.5. What is the impact of the errors inserted into the projects by the maintainers with respect to maintenance effort and code changed ?

In this study we distinguished two types of change requests: user and tester change requests. The original content of the release consists of change requests submitted by users. During the implementation of each release some errors may be introduced by the maintenance work. If these errors are caught by the testers, they in turn generate tester change requests, which become part of the same release delivery. The 25 complete releases contained 187 user change requests, which required 138,000 SLOC. The same releases had 101 tester change requests, which required 3600 SLOC. Thus the tester change requests accounted for 35% of the SCRs in the release, but only 2.5% of the SLOC, as is shown in Figure 4.

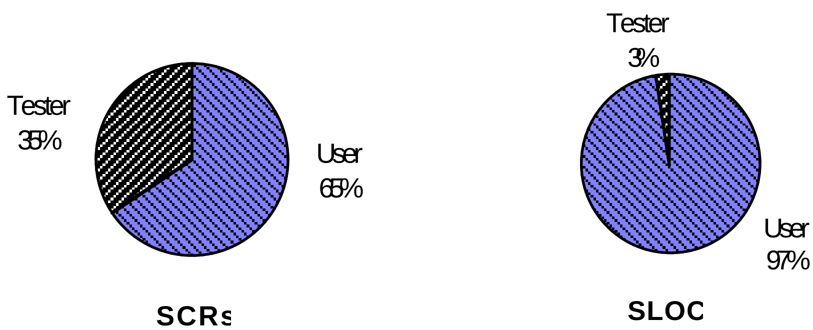

Figure 4: SCR count and SLOC differences between user and tester change requests (for 25 releases)

The effort data associated with individual SCRs is incomplete for releases which began before August 1994 (when the authors revised the WMEF), so the percent of effort associated with tester SCRs is unclear, but the SLOC count suggests that it is a small percentage. In a preliminary attempt to examine the distribution of effort between tester change requests and user change requests, the authors selected 5 releases started and completed between August 1994 and June 1995 (see Figure 5). Since enhancements tend to be larger than error corrections, and since all tester change requests are error corrections, we ignored the enhancements requested by the users (there were no adaptations). In this sample 42% of the error correction SCRs are tester SCRs, but these tester SCRs account for only 27% of the programmer effort associated with these the error correction SCRs in these 5 releases. The number of SLOC added, changed, or deleted for these tester SCRs corresponds to 29% of the total number of SLOC changed, added or deleted for all error correction SCRs.

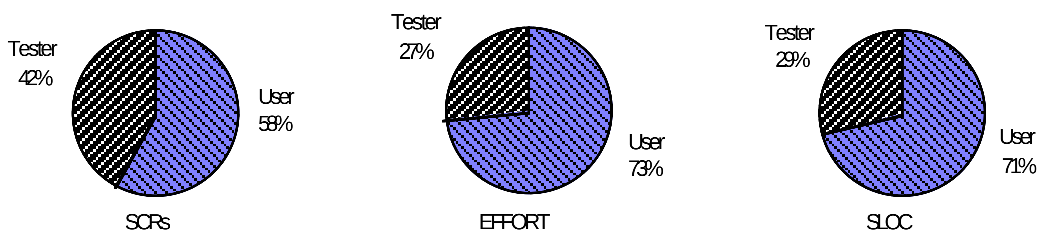

Figure 5: SCR count, Effort, and SLOC differences between 5 completed releases

In order to better comprehend the differences between user and tester SCRs with regard to effort and SLOC we calculated the level of significance of these differences. To do so, we used the Mann-Whitney U non-parametric tests [Hinkle *et al.,* 1995].

We assumed significance at the 0.05 level, i.e., if the p-value is greater than 0.05, then we assume there is no observable difference between tester and user SCRs. The results of these tests as well as other descriptive statistics are provided in Table 1. These statistics are shown for the sake of completeness and also because they help us interpret the results of the analysis in the remainder of this section. In addition, these statistics will facilitate future comparisons of results in similar studies since they will help explain differences in results through differences in statistical distributions.

Descriptive	*User SCRs*			*Tester SCRs*		
Statistics	*SLOC*	*hours*	*Produc.*	*SLOC*	*hours*	*Produc.*
Maximum	300	68	27.27	75	23	16
Minimum	3	2	0.15	4	3	0.19
Median	24	19	1.26	16	16	1.92
Mean	57	26.63	3.50	32.75	13.53	3.76
Std Dev	89.45	22.82	7.94	31.05	7.72	5.21

Mann-Whitney U Test	
Variable	*p-value*
SLOC	0.2069
Hours	0.2637
Productivity	0.3215

Table 1: Descriptive statistics of user and tester SCRs and Mann-Whitney U test results

As this table shows, the mean productivity for user SCRs (3.50) is almost the same as for tester SCRs (3.76). Productivity is defined as the total SLOC added, changed and deleted, divided by the total effort spent to add, change, or delete that SLOC. Based on the results presented in Table 1, we can conclude that there is not a significant difference between the user SCRs as compared to tester SCRs from the perspective of effort, SLOC and productivity (all the p-values are greater than 0.05). Therefore, even though the maintainers have already spent time understanding the code to be modified when the change was first requested, they are not significantly more productive when correcting their own mistakes than they were earlier correcting errors requested by the users. This surprising result is an additional motivation to eliminate errors during the maintenance process. Understanding why tester SCRs are not easier to correct in the current maintenance process may lead to substantial productivity gains.

However, we cannot confirm if this is only a particular situation which happened on these 5 completed releases. We must continue to pursue this analysis in order to verify if these results are valid.

4.4 Release productivity

For this paper, our major concern is how to estimate the cost of subsequent maintenance releases. Planning the next release so as to maximize the increase of functionality and the improvement of quality is vital to successful maintenance management. By analyzing the various relationships between effort and other variables (see Goal 2, we suggest for our environment a predictive model (Goal 3) based upon lines of code per release. By following our procedure (Goals 1, 2 and 3) other organizations can develop their own predictive models, based upon their specific characteristics and the relationships between variables found in their organization. In this section we are interested in answering the following questions related to Goal 3:

- Q3.1. What is the productivity model for the 3 different types of maintenance releases (i.e., enhancement, error correction, and mixture) within the SEL?
- Q3.2. Does a constant amount of overhead exist for any type of maintenance release?

Evaluating the data available on 25 completed maintenance releases within the SEL environment, provided insight into potentially different kinds of maintenance releases. In attempting to develop a cost model for software maintenance releases, we first plotted the size of maintenance releases (measured in SLOC added, changed, and deleted) against the total effort expended on the release. Initial evaluation of this data (by visual inspection) showed that the data seemed to break into 4 different groups.

One group of 4 releases had very high productivity. In trying to find some reason to explain why these releases differed from the others, we noted that the average ratio of units *added* versus *changed* for these 4 releases (1.4) was much higher than for the other 21 releases (0.1). Were the added units primarily reused units (either verbatim or with modification), rather than newly written units, we might assume that the high productivity of these 4 releases was due to their high reuse. But the source of the units added to these 4 releases was not consistent. Sometimes the added units were predominantly borrowed from other projects and reused with modification. But other times the added units were predominantly newly written units. In the latter case, reuse is not the answer.

The answer may be the header, PDL*, comment, and blank lines which SLOC includes in its definition. Newly written units typically contain a high percentage of such lines, but older units—and the maintenance changes made to them—often include a much smaller percentage of such lines. The older units often do not have PDL, so PDL is often not updated when the code is changed. Although more study is needed verify this hypothesis, this reinforces the need to have a thorough knowledge of the process and products in order to interpret the data and build accurate models.

Dismissing these four releases as unusual, we continued to evaluate the remaining 21 releases. Based on an inspection of the data, we developed a scheme for characterizing the other 3 kinds of releases. The three groups seemed to be divided into those releases that were primarily made up of enhancements, those made up primarily of error corrections, and those that fell into neither of these two categories. The scheme used to divide the releases is based on the percentage of change requests within the release that were enhancements or corrections and the percentage of SLOC that

* In the FDD pseudo code, referred to as Program Design Language (PDL), is included in the source code file.

was added, changed, or deleted as a result of enhancements or corrections. Two criteria are established for testing the release type:

- Criterion 1 - (Percentage of Change Requests that are enhancements > 80) or (percentage of SLOC due to enhancements > 80)

- Criterion 2 - (Percentage of Change Requests that are corrections > 80) or (percentage of SLOC due to corrections > 80)

Release type was then determined based on the following test:

```
If (criterion 1) and Not (criterion 2) then Release type = Enhancement
Elseif (Criterion 2) and Not (Criterion 1) the Release type = Correction
Else Release type = mixed
endif
```

This test subdivided the remaining 21 releases into 14 enhancement releases, 3 correction releases, and 4 mixed releases.

The major result of this study is the development of a predictive cost model for maintenance releases that are primarily composed of enhancements. Figure 6 shows the results of a standard linear regression of total release effort versus total lines of code added, changed, and deleted. This model has a coefficient of determination (R^2) of 0.75, which is statistically significant at the 0.00006 level. By estimating the size of a release, an effort estimate can be determined. The equation for the line fit is:

```
Effort in hours = (0.36 * SLOC) + 1040
```

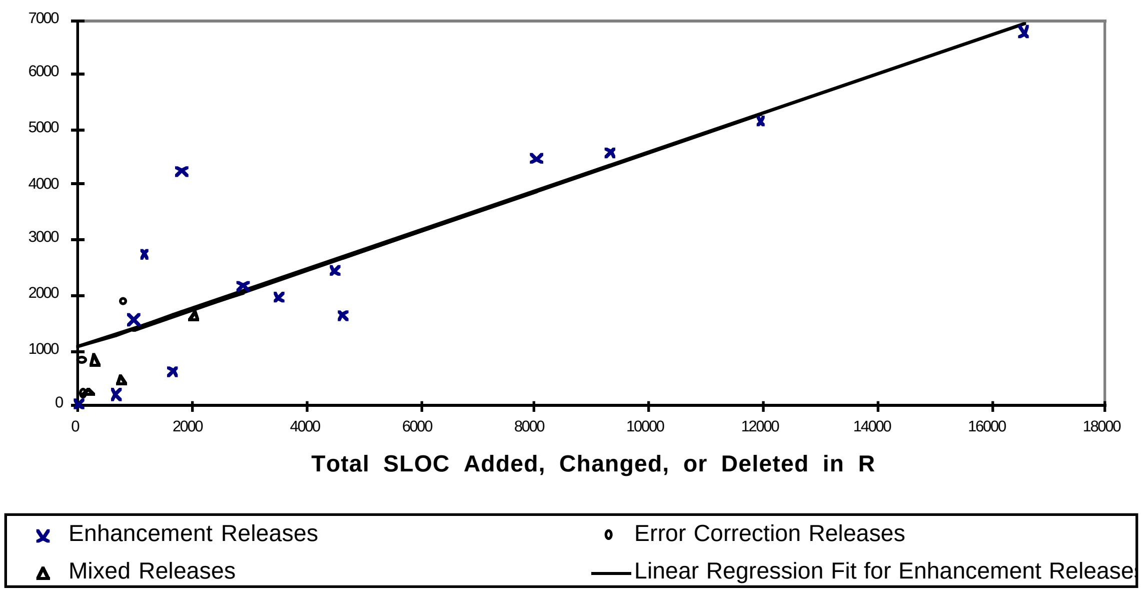

Figure 6. Linear Regression Results for Enhancement Releases

Any maintenance release will have some overhead. It is likely that this overhead stems partly from regression testing and comprehension activities which are somewhat independent from the size of the change. The y-intercept of 1040 hours seems to imply that there is an average release overhead of approximately 1040 hours for enhancement releases in the FDD.

The number of data points for error correction releases and mixed releases makes development of accurate models for them difficult. More data points will be needed to determine if similarly accurate models can be developed. The preliminary data suggests, however, that the productivity for error correction releases and mixed releases is significantly lower than for enhancement releases. This suggests that error corrections are less productive—in terms of SLOC per hour—than are enhancements. The error correction releases and mixed releases tend to be smaller than most of the enhancement releases. If the above observations are true, it may be wise to try to avoid scheduling small error correction releases. Instead the manager should try, when possible, to package small error corrections in a release with larger enhancements. If the enhancements require making changes to the same units or group of units as required by the error corrections, then the savings would likely be larger still. (Of course, with urgent error corrections one does not have the luxury of delaying error corrections until such an opportunity comes along.)

5. Limitations of the data collection and lessons learned

During this research effort, many valuable lessons were learned. These lessons can be divided into general results for studying maintenance and results for data collection.

In the area of lessons learned for studying maintenance, the following statements can be made:

- An overall understanding of the maintenance process and the maintenance environment is crucial to any maintenance study. The combination of qualitative understanding with quantitative understanding has been invaluable. The qualitative understanding helped to drive and improve the data collection process.

- Understanding the environment provides valuable context for the data analysis. Without a thorough understanding of the environment four outlier enhancement releases might not have been recognized as a distinct subset.

In the area of lessons learned on data collection:

- Recognize the limitations of the data and work within those limitations. Data collection by its nature is inexact. Researchers must work within the limits of the data and recognize that the conclusions are only as valid as the data. Qualitative evidence (i.e., structured interviews, analysis of products and process documentation) should be actively used to gain more confidence in the results.

- Assuring the quality of the data collected is a difficult task. The following paragraphs describe the quality assurance procedures and analysis of the quality of the data for this study.

The SCRs are tracked very effectively by the FDD configuration management (CM) team. Their logging and tracking database provided a thorough check on release contents. By comparing the contents of the SEL database with the CM database, we were able to identify any SCRs missing from the SEL database. Copies of these missing SCRs were then acquired from the CM team and entered into the SEL database. Thus, in general, the release contents were characterized to a high level of confidence.

Two minor problems were encountered with the SCR data. First, from talks with maintainers it was learned that the maintainer does not always agree with the change type specified by the user or tester. The user may call a change request an error correction, whereas the maintainer might judge it to be an enhancement. This is not thought to occur in many cases.

Secondly, during the course of the study we learned that not all maintainers were using the same definition in reporting lines of code added, changed, or deleted. The SEL usually uses source lines of code (SLOC), which includes all PDL lines, comment lines, and blank lines, as well as regular lines of code. Most maintainers were using this definition for lines of code on the SCR form. In addition to SLOC, however, the FDD sometimes reports lines of code without counting any PDL, comments, or blanks. The authors learned that some maintainers had been reporting this number on their SCR forms. Luckily most cases were confined to a single project and a single release. For this release the task leader supplied accurate totals of SLOC added, changed, and deleted.

The tracking of weekly effort is not nearly as thorough and rigorous as the tracking of SCRs. No formal audit process exists to assure that all personnel are submitting WMEFs each week they work on a project. Many managers do try to assure that their personnel submit the forms, but the process is not guaranteed.

Still we feel confident that the effort data is reasonably complete and accurate. When possible, data validation has been done with the WMEF data. For example, in some cases, we found that SCRs had been submitted (after the revised WMEF went into effect) but that no maintainer had listed this SCR on his WMEF. The maintainers who worked on these SCRs were then identified and were required to revise their WMEFs.

6. Conclusions and Future Directions

In this paper, we described descriptive models of a software maintenance environment and an incremental approach for the construction of release productivity models for that environment. The former type of models helped us understand better how and why effort is spent across releases while raising new process improvement issues. The latter type of models helped us provide management tools for maintenance task leaders. In order to validate our approach, a case study was conducted at the NASA Software Engineering Laboratory, where we showed the feasibility of building such models. In addition, we derived a set of lessons learned about our maintenance process which allowed us to propose concrete improvement steps. Based on these results, some of the many issues that should be further investigated are discussed below.

As more releases are completed, predictive models for the other categories of releases can be developed. Having cost models for all three types of releases, along with an understanding of the outlier subset of high productivity releases, would complete the cost modeling area of our study. Good cost models for the other types of releases might not be obtainable, but further understanding of the overhead of a release might give better guidance on release content.

In addition to the current model, there is a need for an effort prediction model at the change level. This would help the maintainers perform cost/benefit analysis of the change requests and thereby better determine the release content within budget constraints.

The suite of predictive models can also be expanded to include reliability. We would like to be able to predict, for example, the number of errors uncovered during each maintenance release. Such information will lead to more guidance on release content, and to a better understanding of the release testing process.

Acknowledgment

We want to thank Roseanne Tesoriero for her valuable suggestions that helped us improve both the content and the form of this paper.

References

[Abran & Nguyenkim 1991] Abran, A. and Nguyenkim, H. "Analysis of Maintenance Work Categories Through Measurement," *Proc. Conf. on Software Maintenance 1991*, Sorrento, Italy, pp. 104–113.

[Basili & Rombach 1988] Basili, V. R. and D. Rombach. "The TAME Project: Towards Improvement-Oriented Software Environments," *IEEE Trans. on Software Engineering*, 14 (6), June 1988, pp. 758-773.

[Briand *et al* 1994] Briand, L., V. R. Basili, Y.-M. Kim and D. Squier. "A Change Analysis Process to Characterize Software Maintenance Projects," *Proc. Int'l. Conf. on Software Maintenance*, Victoria, B. C., Canada, pp. 38–49.

[Briand *et al* 1995] Briand, L., W. Melo, C. Seaman, and V. Basili. "Characterizing and Assessing a Large-Scale Software Maintenance Organization," *Proc. 17th Int'l. Conf. on Software Engineering,* Seattle, WA, pp. 133-143.

[Haziza *et al* 1992] Haziza, M.., J. F. Voidrot, E. Minor, L. Pofelski and S. Blazy. "Software Maintenance: An Analysis of Industrial Needs and Constraints," *Proc. Conf. on Software Maintenance 1992*, Orlando, Florida, pp. 18-26.

[Harrison & Cook 1990] Harrison, W. and C. Cook. "Insights on Improving the Maintenance Process Through Software Measurement," *Proc. Conf. on Software Maintenance 1990*, San Diego, CA, pp. 37–45.

[Hinkle *et al* 1995] Hinkle, D. E., W. Wiersma and S. G. Jurs. *Applied Statistics for the Behavioral Sciences*, Boston: Houghton Mifflin, 1995.

[McGarry *et al* 1994] McGarry, F., G. Page, V. R. Basili, and M. Zelkowitz. *An Overview of the Software Engineering Laboratory*, SEL-94-005, December 1994.

[Paulk, *et al*., 1993] Paulk, M., B. Curtis, M-B Chrissis, C. Weber. "Capability Maturity Model, Version 1.1," *IEEE Software,* July 1993, pp. 18-27.

[Pigoski & Nelson 1994] Pigoski, T. M. and L. E. Nelson. "Software Maintenance Metrics: A Case Study," *Proc. Int'l. Conf. on Software Maintenance*, Victoria, B. C., Canada, pp. 392–401.

[Rombach *et al* 1992] Rombach, H., B. Ulery and J. Valett. "Toward Full Life Cycle Control: Adding Maintenance Measurement to the SEL," *J. Systems and Software*, Nov. 1992, pp. 125-138.

[Schneidewind 1994] Schneidewind, N. "A Methodology for Software Quality: Metrics for Maintenance." Tutorial presented at the *Int'l. Conf. on Software Maintenance*, Victoria, B. C., Canada, 1994.

[Valett *et al* 1994] Valett, J., S. Condon, L. Briand, Y.-M. Kim and V. Basili. "Building an Experience Factory for Maintenance," *Proc. 19th Annual Software Eng. Workshop,* NASA Goddard Space Flight Center, November 1994.

[Waligora *et al* 1995] Waligora, S., J. Bailey and M. Stark. *Impact of ADA and Object-Oriented Design in the Flight Dynamics Division at Goddard Space Flight Center*, SEL-95-001, 1995.