Evaluating Software Development by Analysis of Changes: Some Data from the Software Engineering Laboratory

DAVID M. WEISS AND VICTOR R. BASILI, MEMBER, IEEE

Abstract—An effective data collection methodology for evaluating software development methodologies was applied to five different software development projects. Results and data from three of the projects are presented. Goals of the data collection included characterizing changes, errors, projects, and programmers, identifying effective error detection and correction techniques, and investigating ripple effects.

The data collected consisted of changes (including error corrections) made to the software after code was written and baseline, but before testing began. Data collection and validation were concurrent with software development. Changes reported were verified by interviews with programmers. Analysis of the data showed patterns that were used in satisfying the goals of the data collection. Some of the results are summarized in the following.

1) Error corrections aside, the most frequent type of change was an unplanned design modification.

2) The most common type of error was one made in the design or implementation of a single component of the system. Incorrect requirements, misunderstandings of functional specifications, interfaces, support software and hardware, and languages and compilers were generally not significant sources of errors.

3) Despite a significant number of requirements changes imposed on some projects, there was no corresponding increase in frequency of requirements misunderstandings.

4) More than 75 percent of all changes took a day or less to make.

5) Changes tended to be localized with respect to individual components but localized with respect to subsystems.

6) Relatively few changes resulted in errors. Relatively few errors required more than one attempt at correction.

7) Most errors were detected by executing the program. The cause of most errors was found by reading code. Support facilities and techniques such as traces, dumps, cross-reference and attribute listings, and program proving were rarely used.

Index Terms—Software change analysis, software change data, software errors, software measurement.

I. INTRODUCTION

In previous and companion papers [1]-[4] we have discussed how to obtain valid data that may be used to evaluate software development methodologies in a production environment. Briefly the methodology consists of the following five elements.

1) Identify goals. The goals of the data collection effort are defined before any data collection begins. We often relate them to how well the goals for a product or process are met.

2) Determine questions of interest from the goals. Specific questions, derived from the goals, are used to sharpen the goals and define the data to be collected. Answering the questions derived from each goal satisfies the goal.

3) Develop a data collection form. The data collection form used is tailored to the product or process being studied and to the questions of interest.

4) Develop data collection procedures. Data collection is easiest when the data collection procedures are part of normal configuration control procedures.

5) Validate and analyze the data. Reviews and analyses of the data are concurrent with software development. Validation includes examining completed data collection forms for completeness and consistency. Where necessary, interviews with the person(s) supplying the data are conducted.

The purpose of this paper is to present the results from such an evaluation. The data presented here were collected as part of the studies conducted by NASA's Software Engineering Laboratory [5]. In this section we present an overview of the SEL and the projects analyzed for this paper. Section II describes the application of the methodology described in the
foregoing to the SEL environment. Section III presents the results of the data analysis. The analysis and data are contained in the Appendix. Section IV contains conclusions about the SEL environment and some observations on the application of the experimental methodology.

**Overview of the Projects Studied**

The methodology described in [1] was used to study five projects in two different environments: a research group at the Naval Research Laboratory (NRL), and a NASA software production environment at Goddard Space Flight Center (GSFC). The NRL studies have been previously presented [2], [3], [6], [7] and will not be discussed further here. A brief description of the NASA projects follows.

The Software Engineering Laboratory

The Software Engineering Laboratory (SEL) is a NASA sponsored project to investigate the software development process, based at Goddard Space Flight Center (GSFC). A number of different software development projects are being studied as part of the SEL investigations [4], [5], [8]. Studies of changes made to the software as it is being developed constitute one part of those investigations.

Typical projects studied by the SEL are medium size Fortran programs that compute the orientation (known as attitude) of unmanned spacecraft, based on data obtained from on-board sensors. Attitude solutions are displayed to the user of the program interactively on CRT terminals. Because the basic functions of these attitude determination programs tend to change slowly with time, large amounts of design and sometimes code are often reused from one program to the next. The programs range in size from about 20,000 to about 120,000 lines of source code. They include subsystems to perform such functions as reading and decoding spacecraft telemetry data, filtering sensor data, computing attitude solutions based on the sensor data, and providing an (interactive) interface to the user.

Development is done by contractor personnel in a “production” environment, and is often separated into two distinct stages. The first stage is a high-level design stage. The system to be developed is organized into subsystems, and then further subdivided. Each subsystem generally performs a major system function, such as processing telemetry data. The result of the first stage is a tree chart showing the functional structure of the subsystem, in some cases down to the subroutine level, a system functional specification describing, in English, the functions of the system, and decisions as to what software may be reused from other systems. For the purposes of the SEL, each named entity in the system resulting from this process is called a component.

The second stage consists of completing the development of the system. Different components are assigned to (teams of) programmers, who write, debug, test, and integrate the software. Before delivery, the software must pass a formal acceptance test. On some projects, programmers produce an intermediate specification between the functional specifications produced as part of the first stage and the code. Some projects produce pseudocode specifications for individual subroutines before coding them in Fortran. During the period of time that the SEL has been in existence, a structured Fortran preprocessor has come into general use.

The principal design goal of the major SEL projects is to produce a working system in time for a spacecraft launch. In addition, a continuing NASA goal is introducing improved techniques into its software development process. Results from SEL studies of three different NASA projects, denoted SEL1, SEL2, and SEL3, are included here.

II. APPLICATION OF THE EXPERIMENTAL PROCEDURE

A complete list of goals, as described in [1], for the SEL projects is shown in Table I. The analysis presented in this paper will only be concerned with the first six goals from the list. Table II shows the associated questions of interest that were analyzed to produce the results for this paper. Complete lists of goals, questions of interest, and data categories are shown in [9]. The SEL studies represent a full-scale implementation of the data collection methodology in a software production environment. Because the SEL environment is not primarily devoted to developing and proving new methodologies, the emphasis is more on investigating the software development environment than in a study such as [3].

**SEL Goals**

Since the primary emphasis in SEL projects is not on developing and proving new methodologies, the data collection goals are generally methodology-independent. Nevertheless, many of the projects do use recently-developed software engineering technology with a view towards evaluating the technology in the NASA/GSFC environment. (An example is program design language, used in several SEL projects.)

**SEL Questions of Interest**

Since the software was produced in a production environment with stringent deadlines, it was desirable to minimize the overhead involved in collecting and validating data. Because there were no design goals with respect to the use of particular methodologies, questions relating to the success of particular methodologies were generally not considered.

**SEL Data Categories**

Selection of the data categories was based on acquiring the data needed to answer the questions of interest, on maintaining a reasonably small set of subcategories for convenience in collecting and interpreting the data, and on subjective estimates of the uniformity of the data distribution across the subcategories.

The “catch-all” category “other” has been inserted for all changes that will not fit one of the other categories. If the categories selected agree well with the actual change distribution across the subcategories, few errors will fall into the other subcategory. (The reverse situation is not necessarily a sign of a poorly designed categorization scheme; the “other” changes may provide the most insight into the development process.)

**Data Collection, Validation, and Analysis**

Formal procedures used for data collection and validation are described in [1], as is the data collection form.
TABLE I
DATA COLLECTION GOALS FOR THE SEL PROJECTS

1) Characterize changes (especially in ways that permit comparisons across projects and environments).
2) Characterize errors (especially in ways that permit comparisons across projects and environments).
3) Identify effective techniques for detecting errors.
4) Investigate the "ripple" effect, i.e., do most errors require more than one attempt at correction or result in changes distributed over several different components of the system?
5) Characterize projects.
6) Find factors that have significant effects on types and distributions of errors.
7) Evaluate effectiveness of methodologies in NASA/GSFC environments.
8) Identify effective techniques for obtaining the information needed to correct errors.
9) Suggest ways of improving NASA/GSFC software development practices.
10) Verify that concurrent data validation is needed.
11) Characterize programmers.
12) Identify good measures of correctness.

TABLE II
QUESTIONS OF INTEREST

1) What was the distribution of changes according to the reason for the change? Reasons were considered to be one of the following:
   a) a change in requirements or specifications;
   b) a change in design;
   c) a change in hardware environment (e.g., a new piece of hardware added to the system to be used by the program);
   d) a change in software environment (e.g., a new version of the Fortran compiler);
   e) an optimization;
   f) other.
2) What was the distribution of changes across system components?
3) What was the distribution of effort required to design changes?
   For error corrections, the effort required to design the change was assumed to be the same as the effort required to understand the error and propose a correction.
4) What was the distribution of errors according to the misunderstandings that caused them?
5) What was the distribution of effort required to correct errors?
6) How many errors were the result of software changes?
7) What was the distribution of errors across error detection techniques?
8) What was the number of attempted error corrections per error?

Answering Questions of Interest

The questions of interest are answered by presenting and analyzing the data distribution(s) associated with each question. Because of space limitations, answers to the individual questions, and tables and histograms used in the data analysis have been included in the Appendix.

Overview of the Data

Table III contains an overview of the data collected and a summary of information about the projects. Changes are divided into two categories: error corrections and modifications. Modifications are changes made for purposes other than error correction. Values of parameters often thought to characterize software development projects are also shown in Table III.

III. INTERPRETATIONS

The research methodology permits at least one quite straightforward way of interpreting the data: using the distributions to answer the questions of interest, thereby satisfying the goals of the study. One may also compare distributions across different projects, where appropriate, and look for common characteristics. Both of these processes lead to new goals and questions, some of which may be answerable with the available data, and some requiring new studies. Examples of both will be presented here.

Table IV shows, for each goal analyzed in this paper, the corresponding questions of interest. Where the same question(s) are used to satisfy several goals, the goals are listed together.

Readers interested in the detailed analysis of the data distributions used to answer the questions of interest are referred to the Appendix. Based on the analysis in the Appendix, we present here a summary of results for the goals.

In the following sections each goal is satisfied by presenting conclusions based on the answers to the questions corresponding to the goal. A short description of each goal precedes its discussion.

Goal: Characterize Changes

All three projects operated in a stable environment where there were few changes to the support software and hardware; none of them made many changes for the purpose of adding or deleting debug code. The results support the view that the SEL designers have organized their systems so that, for purposes of redevelopment, most changes are confined to a few subsystems.

One way that the projects clearly differ is in their reasons for making unplanned design changes. Some spend a great deal of time on optimization and proving the services the system offered to its users, others on attempting to improve the clarity of the code and its documentation. It is interesting to note that SEL2 and SEL3, whose programmers had different reasons for making unplanned design modifications, had the same task leader and some of the same staff.

Coupled with the effort and the component-wise change analyses, the results suggest that most unplanned design modifications are small and only involve one component of the system. Several explanations are possible; either the pro-
TABLE IV
RELATIONSHIP BETWEEN GOALS AND QUESTIONS

<table>
<thead>
<tr>
<th>Goal: Characterize changes.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Questions:</td>
</tr>
<tr>
<td>What was the distribution of modification according to the reason for the modification?</td>
</tr>
<tr>
<td>What was the distribution of changes across system components?</td>
</tr>
<tr>
<td>What was the distribution of effort required to design changes?</td>
</tr>
<tr>
<td>Goal: Characterize errors.</td>
</tr>
<tr>
<td>Questions:</td>
</tr>
<tr>
<td>What was the distribution of errors according to the misunderstandings that caused them?</td>
</tr>
<tr>
<td>What was the distribution of effort required to correct errors?</td>
</tr>
<tr>
<td>How many errors were the result of software changes?</td>
</tr>
</tbody>
</table>

Grammars act as "filters," rejecting unplanned modifications that are not easy to make, or reasons for modifying the design are not characteristic of the programmers, but rather of some external source.

Some Conclusions Concerning Characterization of Changes

Although it is tempting to try to characterize a "typical" modification, there is too much variability in the sources of modifications for the different projects to do so safely. The sources for most modifications fall into one of a small number of subcategories, such as requirements modifications, planned enhancements, improvements of clarity, improvements of user services, and optimizations. The distributions over these categories distinguish one project from another.

The SEL projects are all similar with respect to the effort required to modify the programs; most changes and modifications take a day or less to make. Furthermore, although the changes tend to be nonlocal with respect to individual components (most components that are changed are only changed once or twice), they are localized with respect to subsystems, i.e., the majority of changes are made in one or two subsystems.

Goal: Characterize Errors

From the answers to the questions we may conclude that the SEL programmers tend to spend their time finding and correcting many "small" errors made while designing or implementing single routines, rather than struggling with a few "large" errors, or trying to understand requirements or interfaces.

All the SEL projects handled changes with little trouble; relatively few errors were the result of a change to the software. Requirements appear to be well enough understood that changes to them can be handled with little trouble. Interfaces, often considered to be a major source of errors, do not seem especially troublesome. There is some indication that the interface and requirements understandings that do occur are more difficult to correct than others. However, the small number of errors involved makes it dangerous to draw such a conclusion.

We believe there are two factors that explain the shape of the error distributions and their similarity across projects.

1) The SEL projects all have the same application. They are essentially redevelopments, each using the same overall design and often much of the same code as previous projects. Although new individual programmers may be used from one project to the next, the same people do the top level design. Having found a successful design, they reuse it.

2) The SEL projects used programmers who were familiar with the language they were using, and both were developed in a stable environment, i.e., there were few changes in support hardware or software.

Some Conclusions Concerning Error Characterization

Based on the foregoing analysis, one might characterize a "typical" error as one that occurs in the design or implementation of a single component, is easy to correct, and whose cause is easy to find.

Goal: Identify Effective Error Detection Techniques

Executing the program was the most successful means for detecting errors. The distributions show what might be called a traditional approach to error detection: either test runs or a programmer reading over her own code.

Goal: Investigate the Ripple Effect

There is nothing in the data to suggest a ripple effect of any significance. The lack of such an effect may be the result of the SEL experience with the application. It may also be a result of monitoring the projects primarily through the development phase. Continued monitoring throughout the project lifetime might reveal such an effect as the software undergoes further change.

Goal: Characterize Projects

Inspection of the change distributions shows that, despite the similarities in application, environment, and personnel, there are distinct differences among SEL projects. Some projects, notably SEL3, seem to have considerably less trouble in the development phase than others.

There are two possible explanations: 1) the SEL3 developers did a better job in producing correct software, or 2) the SEL3 system was not subjected to a thorough inspection for errors. Discussions with SEL personnel indicate that SEL3 was no more error prone in operation than either SEL1 or SEL2, suggesting that 1) is the correct explanation.

Examination of the data shows that it is risky to characterize a project with a single parameter or distribution. Furthermore, it is difficult to predict the effect that a particular project characteristic will have on any particular change distribu-
tion. We can identify variations in distributions that seem to
distinguish some projects from others, and use the distinguish-
ing distributions as the basis for more detailed experiments.
A list of candidate distributions follows.

**Sources of Modifications:** The sources of modifications dis-
tributions all show their strongest peaks in the same places,
but have secondary peaks in different places. These secondary
peaks may be used to distinguish among projects. SEL2 and
SEL3 both show strong peaks in requirements changes. SEL1
and SEL3 both show peaks in the planned enhancement cate-
gory. SEL1 has a much stronger peak in the design category
than either of the others.

**Sources of Nonclerical Errors:** All projects show a strong
peak in the same place in the sources of nonclerical errors dis-
tributions. SEL3 may be distinguished from the other SEL
projects by its secondary peak in the "Design MultiComp"
category. SEL1 shows a somewhat stronger peak in the "Fin
Spec" category than the other projects.

**Effort to Design Change:** All SEL projects have design effort
distributions of about the same shape. The only variation is in
the proportion of the distribution contained in each category.
SEL1 shows a considerably stronger peak in the Easy category
than any of the other projects.

**Frequency Distribution of Changes:** The SEL1 and SEL2
component change frequency distributions show a generally
similar shape except for the first category.

**Characteristics of the SEL Projects**

By analyzing the appropriate distributions, the SEL projects
may be characterized as follows.

1) Software production takes place in an environment stable
with respect to hardware and software support.

2) Programs are produced by making many small changes to
a set of initial code. A significant number (40 percent or
more) of these changes are error corrections. Most of the
changes are not planned in advance. Relatively few of them
result in errors.

3) Most changes that are not error corrections are design
changes made for the purposes of optimization, improving
the clarity and maintainability of the code, improving the
documentation (including comments in the code), or improv-
ing the services provided to the user by the program.

4) Most errors occur in the design or implementation of one
component of the system, and are easy to find and easy to
correct. Errors are usually corrected on the first try.

5) Although most changes are concentrated in two or three
subsystems, few individual components are changed more than
three or four times.

6) Although a project may have relatively many require-
ments changes, these changes do not constitute a major source
of errors. Interface errors are also not especially troublesome.

**Goal: Find Factors That Significantly Affect Distributions
of Errors**

It is not possible in these studies to isolate particular factors
and examine their effect on the various error distributions.
Nevertheless, it was expected that patterns of influence would
be visible. One expected pattern was that the distribution of
sources of modifications would affect the distribution of
sources of errors, e.g., the greater the number of requirements
changes, the greater the number of requirements errors. This
expectation was not confirmed; the sources of errors seem to
be relatively independent of the sources of modifications.

Other factors that were expected to contribute heavily as
error sources, but apparently did not, include the software
development environment, the programming language used,
misunderstandings of interfaces, project size, and misunder-
standings of specifications.

The error distributions for the SEL projects indicate that the
single most important factor is the method used by the indi-
vidual programmer in designing and coding individual routines.
More detailed studies of individual programmer techniques in
the SEL environment might indicate particular methodological
weaknesses.

Generalization of these results to other environments may
not be possible. In the SEL projects certain circumstances
may have acted to decrease the effects of certain factors. SEL
experience with the application, and the adaptation of pre-
vious designs in the development of new systems are in this
category.

**IV. Conclusions and Summary**

The SEL data collection projects showed that it was feasible
to collect and validate data on all changes concurrently with
software development. (A companion paper [1] shows that it
was necessary to perform validation by means of developer
interviews.) The data collected permit the following charac-
terization of the SEL environment, projects, and programmers.

1) Error corrections aside, the most frequent type of change
is an unplanned design modification. Such modifications are
usually made for one of the following reasons:

   a) to optimize the program,

   b) to improve the services the program offers to its users, or

   c) to improve the clarity and maintainability of the pro-
      gram and its documentation.

2) The most common type of error is one made in the de-
sign or implementation of a single component of the system.
Incorrect requirements and misunderstandings of functional
specifications, interfaces, support software and hardware, and
languages and compilers are generally not significant sources
of errors.

3) Despite a significant number of requirements changes
imposed on some projects, there is no corresponding increase
in frequency of requirements misunderstandings. A possible
explanation is that the developers understand the application
sufficiently well that their design is easily adaptable to most
requirements changes, i.e., they know what kinds of changes
to expect and have designed for them.

4) More than 75 percent of all changes take a day or less to
make. Most programmers apparently spend their time making
many small changes to their programs, rather than few large
ones.

5) Changes tend to be nonlocalized with respect to individ-
ual components (most components that are changed are only
changed once or twice), but localized with respect to sub-
systems (the majority of changes are made in one or two subystems).

6) Relatively few changes result in errors. Relatively few
errors require more than one attempt at correction.

7) Most errors are detected by executing the program. The
cause of most errors is found by reading code. Techniques
and support facilities such as program proving, cross-reference
and attribute lists, and dumps (that were once so popular
that papers, such as [10], were published on how to read
them) are rarely used.

Opportunities Missed
The data presented here and in [2], [3], [6], and [7] repre-
sent five years of data collection. During that time there was
considerable and continuing consideration given to the ap-
propriate goals and questions of interest. Nonetheless, as data
were analyzed, it became clear that there was information that
was never requested but that would have been useful. An
example is the length of time each error remained in the sys-
tem. Programmers correcting their own errors, which was the
usual case, could supply these data easily. One could then iso-
late errors that were not easily susceptible to detection by pro-
gram execution or code reading. This example underscores
the need for careful planning prior to the start of data collection.

Understanding and Improving the Environment
Knowing the kinds of changes and errors made to the soft-
ware is a key to understanding the software development
environment. Obstacles to change, methods and designs that
facilitate change, implementation difficulties, and troublesome
components are examples of elements highlighted by appro-
priate data collection. The knowledge gained enables one to
work effectively to increase productivity and improve software
quality.

Comparing Environments
In most sciences, valuable information is gained from repeat-
ing experiments, sometimes to confirm new results, other times
to refine them. We believe this should be the case in Com-
puter Science. Although some interesting patterns are exhi-
bited in the SEL data, it would be useful to seek similar trends
in data from other environments. Unfortunately, there exists
little comparable data ([4] is one exception). A primary rea-
son for devising the data collection methodology used here is
to show how comparable data from different environments
may be collected. Common goals, questions of interest, and
data categorization may be used to ensure comparability.

APPENDIX

Answering Questions of Interest
The questions of interest are answered by presenting and ana-
lyzing data from distribution(s) associated with each ques-
tion. For each question there is a short discussion of the asso-
ciated distributions. The main purpose of the discussions is
to point out various features of the distributions that are of
significance in answering the questions. Table V shows the
relation between the questions and the distributions. Only
those questions used to derive conclusions in the body of this
paper are discussed here.

<table>
<thead>
<tr>
<th>TABLE V</th>
<th>FIGURE</th>
<th>TABLES USED IN ANSWERING QUESTIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) What was the</td>
<td>Figs. 3, 5</td>
<td>Table II</td>
</tr>
<tr>
<td>distribution of</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| modifications ac-
| cording to the |       |                                   |
| reason for the |       |                                   |
| modification?   |       |                                   |
| 2) What was the | Figs. 6, 7 | Table III |
| distribution of |       |                                   |
| changes across |       |                                   |
| system componen-
| ts?             |       |                                   |
| 3) What was the | Fig. 8 |                                   |
| distribution of |       |                                   |
| effort required  |       |                                   |
| to design changes? | |                                   |
| 4) What was the | Figs. 1, 9 | Table VI |
| distribution of |       |                                   |
| errors according |       |                                   |
| to the misunderstandings that caused them? | |                                   |
| 5) What was the | Fig. 2 |                                   |
| distribution of |       |                                   |
| effort required |       |                                   |
| to correct errors? | |                                   |
| 6) How many errors |       |                                   |
| were the result of software | |                                   |
| changes?         |       |                                   |
| 7) What was the | Table III |                                   |
| distribution of errors across error detection techniques? | |                                   |
| 8) What was the number of attempted error corrections per error? | |                                   |

One purpose of this research is to provide a set of empiri-
cally derived data that others may use in constructing models
and deriving hypotheses. The data presented here may be so
used. Most of the presentations are in the form of histograms
based on data categorizations previously discussed. The fol-
lowing sections are intended to help the reader understand
the organization and content of the various histograms.

Organization of Data Presentation.
In general, the histograms are organized into figures, with
each figure containing corresponding histograms for all proj-
ects. An example is Fig. 1, which shows the sources of non-
clerical errors for all projects. For some figures, not all proj-
ects are represented, since a particular set of data may not be
relevant or available for some projects.

Labels on histograms are generally mnemonic abbreviations of
descriptions of data categories (e.g., PE means planned en-
hancement). Keys, supplied for nonobvious labels, provide
the complete name for each mnemonic.

Data Categorization
During the data collection period, several improvements
were made to the names of the projects contain more categories than for others. A second result is that there are occasional differences in the names and meanings of similar subcategories for different proj-
ects within a particular figure. Such differences in categoriza-
tion are discussed in the next few sections.

Changes in Measurement Precision
Data categories for some of the projects contain finer data
quantifications than others. An example is the SEL1 and
SEL3 categories shown in Fig. 2, "Effort To Change NonCler-
ical Errors." The SEL3 figure has a larger set of categories
than the SEL1 figure. After analyzing the results of our early
data collection efforts, we realized it was possible to and of
interest to use a finer measure of effort.

Insufficient Subcategorization
As a result of inexperience, some data categories were too
broad, and some too narrow on the early versions of the data
collection forms. As an example, a design change category was
included on the form at one time. So many changes were re-
ported in this category that it was important to subcategorize further. (The next version of the form contained the new subcategories explicitly.) Fig. 3 shows the subcategories for all SEL projects. Conversely, environment changes occurred sufficiently rarely so that it was unnecessary to distinguish between hardware and software environment changes. These categories were merged during data analysis.

The "Unknown" Category

Despite the intensive review and interview process used for validation, there were still cases where it was not possible to categorize certain changes. This occurred most often for the various effort categories when forms were generated. These cases are categorized as unknown in the histograms where they appear.

Fine Distinctions That Can Be Made

For much of the data, the variety of data categorizations, the comments supplied by the programmers, and the information gained from validation permit certain fine distinctions to be drawn during analysis. An example is the distinction among errors affecting more than one component, design errors involving several components, and interface errors. Interface errors may be divided into two classes. The first class consists of incorrect assumptions between modules and routines. An example involved an assumption about initialization. The programmer of one module assumed that it was necessary to invoke an initialization routine from a second module each time he used certain routines from the second module. This assumption was incorrect. The second class consists of errors in using interfaces, where such errors are
not the result of incorrect assumptions. An example is a programmer forgetting to include a parameter in a calling sequence.

Design errors involving several components are errors in the organization of the software into components, including the specifications that describe that organization. Although this category includes many interface errors, it also includes errors that are not interface errors.

Errors affecting more than one component are errors whose corrections require changes to be made in more than one component. These errors may fit any of the categories of misunderstandings, and are not necessarily interface errors.

**Distinctions That Were Too Fine**

For some categories, developers were asked to make fine distinctions in supplying the data. The metric used for measuring difficulty of fixing nonclerical errors (see Fig. 2) is an example. For SEL1 and SEL2, programmers were asked to separate the effort just to design the change from the effort to make the change. This distinction was too fine for the programmers reporting the effort, and during SEL3 data collection just the total effort was requested.

**Comparing Distributions-Arithmetic Considerations**

To convert raw data counts into measures that could be used to compare projects, percentage of changes in a particular category is usually used. As an example, in Fig. 1, values in the distributions are shown as percentages of non clerical errors. Because there are generally large differences in values within any distribution, the values are rounded to whole percents. For each distribution, any category that is nonempty is assigned a nonzero value. As a result, some categories that contain less than 0.5 percent of the distribution are shown as containing 1 percent. (Categories that contain no data do not appear in the distributions.) For no distribution does this make a difference of more than 1 percent in any category. For some distributions, there is a resulting roundoff error.

**Answers to the Questions**

In the following sections we discuss the answers to the questions of interest. Sections are headed by short descriptions of questions.

**Overview of SEL Changes**

There is no question that deals with all changes; modifications and errors are characterized separately. Nevertheless, analysis of the data showed that it was of interest to look at the overall change distributions and compare them across projects.

Fig. 4 shows some interesting differences among the three projects. The proportion of both all errors and of nonclerical errors declines from SEL1 (46 and 47 percent, respectively) through SEL3 (40 and 32 percent, respectively). The SEL3 developers also appear to have been considerably more occupied with making modifications than with correcting nonclerical errors. Various parameters that normalize number of changes and errors with respect to size in terms of effort and lines of code show the same trend. From these distributions and parameters it appears that there are distinct differences among SEL projects, and that some projects seem to have considerably less trouble in the development phase than others.

**What was the Distribution of Modifications According to the Reason for the Modification?**

Modification distributions are shown in Fig. 5. All projects show a strong spike in the design change subcategory. There is considerable variability in several other categories. SEL2 and SEL3 both experienced relatively large numbers of requirements changes. SEL1 and SEL3 both show considerable use of planned enhancements. Similars in the distributions show that all three projects operated in a stable environment, where there were few changes to the support software and hardware, and that none of them
made many changes for the purpose of adding or deleting debug code.

Fig. 3 is an analysis of design modifications only. Again, there is considerable variability in the distributions. SEL1 programmers were considerably concerned with optimization, i.e., improving the efficiency of use of memory and processor time, and improving the services the system offered to its users.

The SEL2 distribution, whose pattern is somewhat less clear because of the large size of the "unknown" category, also shows emphasis on optimization, and, to a considerably lesser degree, on improving user services and the clarity and maintainability of the program and its documentation. In SEL3, the emphasis is reversed; there were relatively few attempts at optimization, but many at improving clarity, maintainability, and documentation. It is interesting to note that SEL3 had the same task leader and some of the same staff as SEL2.

What Was the Distribution of Changes Across System Components?

In other discussions of changes, we view a change as a logical unit, independent of how much code or documentation, or how many components were involved. For purposes of analyzing frequency distributions of changes, we consider the number of changes made to each component. The number of changes made to a component is considered to be the number
of change report forms on which that component is named as being changed. Using this definition of change, Fig. 6 shows the percentage of components that were changed once, twice, etc. As an example, for SEL1, 29 percent of the components were changed once, and 30 percent were changed twice.

The frequency distributions for all the SEL projects show the same pattern: 50 percent or more of the components that were changed were only changed once or twice, and more than 90 percent were changed six times or less.

Fig. 7 shows the patterns of subsystems that are changed most often. (The distributions are obtained by grouping the data for the components into subsystems.) It is clear from these distributions that at most two or three of the subsystems receive the most attention.

What Was the Distribution of Effort Required to Design Changes?

Change effort distributions are shown in Fig. 8 which shows the effort for all changes except clerical errors. Examining Fig. 8, one can see that most (more than 75 percent) of changes fall into the easy or medium categories for all SEL projects.

What Was the Distribution of Errors According to the Misunderstandings That Caused Them?

Inspection of the distributions showing sources of nonclerical errors (Fig. 1) shows noteworthy similarities across projects. The distributions all show strong spikes in the same places; it is evident that the major source of errors is in the design and implementation of single components.

Factors such as misunderstandings of requirements and specificifications are minor sources of errors. (Note that Fig. 5 shows significant numbers of requirements changes for projects SEL2 and SEL3. The SEL developers apparently understand their requirements well enough that they can handle changes to them without much trouble.) Interfaces are also a minor error source (Fig. 9).

What Was the Distribution of Effort Required to Correct Errors?

Effort distributions for correcting errors are shown in Fig. 2. (Note that there is a slight difference in the type of effort measured for SEL3 than for SEL1 and SEL2.) As shown by these distributions, most error corrections take little effort. For all
projects, approximately 50 percent or more of the errors were corrected in one hour or less, and more than 85 percent were corrected in one day or less.

_How Many Errors Were the Result of a Software Change?_  
Table III shows that the SEL projects handled changes with little trouble; relatively few errors were the result of a change to the software.

_Was the Distribution of Errors Across Error Detection Techniques?_  
The relative frequency of use of various error detection techniques are shown in Table VI for the SEL projects. While examining the distributions, one must recall that SEL change monitoring did not begin until code was baseline and had already undergone debugging. Otherwise, error messages might rank higher as a detection technique.

Executing the program was the most successful means for detecting errors. The distributions show what might be called a traditional approach to error detection: either test runs, or a programmer reading over her own code.

What _Was the Number of Attempted Error Corrections Per Error?_  
If any of the projects suffers from a ripple effect, we expect to see many errors requiring repeated attempts at correction, and many changes each resulting in several errors. As can be seen from Table III, both of these effects appear quite small. The worst case is about 6 percent of the changes resulting in errors (SEL2). The errors resulting from change for the worst case (SEL2) comprised 14 percent of all errors. Finally, very few errors required more than one attempt at correction (these are a subset of the errors resulting from change, since each attempted correction is considered to be a change).

**_ACKNOWLEDGMENT_**  
The authors thank the many people at NASA/GSFC and Computer Sciences Corporation who filled out forms and submitted to interviews, especially J. Grondalski and Dr. G. Page, and the librarians, especially S. DePriest.

We thank Dr. J. Gannon, Dr. R. Meltzer, F. McGarry, Dr. G. Page, Dr. D. Parmas, Dr. J. Shore, and Dr. M. Zelkowitz for their many helpful suggestions.
<table>
<thead>
<tr>
<th>Error Detection Activities</th>
<th>Error First Detected by</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEL1</td>
<td>SEL2</td>
</tr>
<tr>
<td>Test Runs</td>
<td>128</td>
</tr>
<tr>
<td>Preacceptance Test Runs</td>
<td>**</td>
</tr>
<tr>
<td>Acceptance Testing</td>
<td>**</td>
</tr>
<tr>
<td>Code Reading by Programmer</td>
<td>59</td>
</tr>
<tr>
<td>Code Reading by Other Person</td>
<td>21</td>
</tr>
<tr>
<td>Reading Documentation</td>
<td>1</td>
</tr>
<tr>
<td>Trace</td>
<td>4</td>
</tr>
<tr>
<td>Dump</td>
<td>1</td>
</tr>
<tr>
<td>Cross Reference or Attribute List</td>
<td>6</td>
</tr>
<tr>
<td>Special Debug Code</td>
<td>11</td>
</tr>
<tr>
<td>General/System Error Message</td>
<td>3</td>
</tr>
<tr>
<td>Project Specific Error Messages</td>
<td>2</td>
</tr>
<tr>
<td>Inspection of output</td>
<td>12</td>
</tr>
<tr>
<td>Proof Technique</td>
<td>4</td>
</tr>
</tbody>
</table>

*This category was subdivided into the categories Preacceptance Testing for SEL3.

**This category was not used for SEL1 and SEL2. See preceding note.

Deserving of special mention is F. McGarry, who had sufficient foresight and confidence to sponsor much of this work and to offer his projects for study.

Finally, we thank C. Hinson, T. Lewis, and the other programmers who worked on the programs used to display the data.

REFERENCES


David M. Weiss received the B.S. degree in mathematics in 1964 from Union College and the M.S. and Ph.D. degrees in computer science from the University of Maryland, College Park, in 1974 and 1981, respectively.

Since 1975 he has been on the research staff at the Naval Research Laboratory, Washington, DC, currently with the Computer Science and Systems Branch. His research interests are in software engineering, software change analysis, and formal specification. He is a member of the software cost reduction project whose purpose is to provide a well-engineered model of a complex real-time system.

Victor R. Basili (M'83) received the Ph.D. degree in computer science from the University of Texas at Austin.

He is currently a Professor and Chairman of the Department of Computer Science at the University of Maryland, College Park, where he has been since 1970. He has been involved in the design and development of several software projects, including the SIMPL family of structured programming languages, and is currently involved in the measurement and evaluation of software development at the NASA/Goddard Space Flight Center. His interests lie in the software development methodology and the quantitative analysis and evaluation of the software development process and product. This includes such specialized areas as cost modeling, error analysis, and complexity. He has consulted for several government agencies and industrial organizations, including IBM, GE, CSC, NRL, NSWIC, and NASA.

Dr. Basili is a member of the Association for Computing Machinery and the IEEE Computer Society. He has been Program Chairman for several conferences and has served on several editorial boards.