Technical Report TR-1236 December 1982
NSG-5123

EVALUATING SOFTWARE DEVELOPMENT BY ANALYSIS OF CHANGES: THE DATA FROM THE SOFTWARE ENGINEERING LABORATORY *

Victor R. Basili
University of Maryland

David M. Weiss
Naval Research Laboratory

*Research supported in part by the National Aeronautics and Space Administration Grant NSG-5123. Computer support provided in part by the facilities of NASA/Goddard Space Flight Center and the Computer Science Center at the University of Maryland.

ABSTRACT

An effective data collection methodology for evaluating software development methodologies was applied to four different software development projects. Goals of the data collection included characterizing changes and errors, characterizing 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 baselined, 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 and 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% of all changes took a day or less to make.

5. Changes tended to be nonlocalized 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.

Evaluating Software Development By Analysis Of Changes: The Data From The Software Engineering Laboratory

Victor R. Basili

University of Maryland

and

David M. Weiss

Naval Research Laboratory

1. Introduction

In previous and companion papers [1, 2, 3, 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. From the goals, specific questions are derived. 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].

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, 6, 3, 7] and will not be further discussed 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 [8, 5]. 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 re-used 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. For the purposes of the SEL, each named entity in the system is called a component. 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.

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 no intermediate specifications between the functional specifications produced as part of the first stage and the code. Some projects produce pseudo-code 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.

2. Application Of The Experimental Procedure

The goals, questions of interest, and data categorizations, as described in [1], for the SEL projects are shown in table 1 and lists 1 and 2. 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.) As a result, the goal "evaluate effectiveness of methodologies" is used, but is not based on specific claims for specific methodologies.

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. Evaluate effectiveness of methodologies in NASA/GSFC environment.
4. Suggest ways of improving NASA/GSFC software development practices.
5. Verify that concurrent data validation is needed.
6. Identify good measures of correctness.
7. Identify effective techniques for detecting errors.
8. Identify effective techniques for obtaining the information needed to correct errors.
9. 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?
10. Characterize projects.
11. Characterize programmers.
12. Find factors that have significant effects on types and distributions of errors.

Table 1. Data Collection Goals for the SEL Projects

---

1. What was the distribution of changes according to the reason for the change and the effect of the change? Reasons were considered to be one of the following:
    a. a change in requirements or specifications,
    b. 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.

    Since a change to any of the items in the preceding list could affect others on the list, the set of items that could be affected by a change were as follows:

    a. requirements or specifications,
    b. design,
    c. the hardware environment,
    d. the software environment,
    e. optimization algorithms and their implementation.

List 1. Questions of Interest

2a. What was the distribution of changes across system components?

2b. For each change, how many components have to be examined in order to make the change?

3. What was the distribution of time required to design changes? For error corrections, the time required to design the change was assumed to be the same as the time required to understand the error and propose a correction.

4. What was the ratio of changes not made to correct an error to error corrections as a function of time during the development cycle?

5. What was the distribution of errors according to the misunderstandings that caused them (and what was the ratio of non-clerical to clerical errors?) ?

6. What was the distribution of effort required to correct errors?

7. What was the distribution of effort to correct errors across misunderstandings causing errors?

8. How many errors were the result of a software change or modification (a modification is a change made for some purpose other than correcting an error)?

9. What was the distribution of errors across error detection techniques?

10. What was the distribution of errors across error correction techniques?

11. What was the number of attempted error corrections per error?

12. What was the distribution of error corrections across project phases?

13. What was the ratio of errors to various measures often associated with with effort and productivity. These measures include

    a. number of developers
    b. number of lines of code
    c. number of machine instructions
    d. number of memory words
    e. number of person-hours
    f. number of work assignments.

14. What was the distribution of errors per person according to the number of people involved?

15. What was the number of errors for projects requiring memory overlays compared to those not requiring overlays?

16. What was the distribution of errors according to programmer?

17. How often must reported change data be corrected as a result of the data validation process?

List 1. Questions of Interest (continued)

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.

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 most tables and histograms used in the data analysis have been included in the Appendix.

Overview Of The Data

Tables 2 and 3 contain, for quick reference, an overview of the data collected and a summary of information about the projects. Tables 4 through 7 contain values of parameters often thought to characterize software development projects.

3. 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.

List 3 shows, for each goal, the corresponding questions of interest. Where the same question(s) are used to satisfy several goals, the goals are listed together.

1. Effort to change. Subcategories:

   a. one hour or less
   b. one hour to one day
   c. one day to three days
   d. more than three days.

2. Cause of change and effect of change. Causes of changes 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,
   d. a change in software environment,
   e. an optimization,
   f. other.

   Since a change to any of the items in the preceding list could affect others on the list, the set of items that could be affected by a change were as follows:

   a. requirements or specifications,
   b. design,
   c. the hardware environment,
   d. the software environment,
   e. optimization algorithms and their implementation.

3. Component changes. This categorization shows, for each component, the number of changes made to the component. There is, accordingly, one subcategory for each component of the system. A similar categorization is used for the number of times each component is examined, i.e. the number of changes that required examination of the component.

4. Result of modification (for error corrections only). Subcategories:

   a. Result of modification not to correct an error, for errors resulting from a program change other than an error correction,
   b. Result of error correction, for errors reslting from a program change made to correct an error (whether a prior correction attempt for the same error, or a correction for some other error),
   c. Not the result of a modification, for errors that are unrelated to program changes.

5. Time to isolate cause (for error corrections only). Subcategories:

   a. one hour or less
   b. one hour to one day
   c. more than one day

List 2. Data Categories

6. Causative misunderstanding. Subcategories:

   a. misunderstanding of requirements
   b. misunderstanding of functional specifications
   c. misunderstanding of other documentation
   d. misunderstanding of design (excluding interface)
      This subcategory was deemed sufficiently interesting to be further subdivided into the following subcategories: misunderstanding of intended use of the erroneous segment/proc/module, misunderstanding of the value or structure of data, and other.
   e. misunderstanding of interface
   f. misunderstanding of programming language, further subdivided into syntax and semantics misunderstandings
   g. misunderstanding of hardware environment
   h. misunderstanding of software environment
   i. clerical error
   j. other

7. Development phase when error occurred. Subcategories:

   a. requirements
   b. functional specifications
   c. design
   d. coding and test
   e. other
   f. can't tell, for situations where the person supplying the information does not know the phase.

8. Method of detection. Subcategories:

   a. test runs
   b. code reading by programmer
   c. code reading by other person
   d. reading documentation
   e. proof technique
   f. trace
   g. dump
   h. cross-reference
   i. attribute list
   j. special debug code
   k. error messages, further subdivided into general error messages, and project specific (i.e. coded especially for this project) error messages
   l. inspection of output
   m. other

List 2. Data Categories (continued)

Project	Number of Changes	Number of Modifications	Number of Errors
SEL1	281	101	180
SEL2	229	110	119
SEL3	760	453	307

Table 2. Overview of Data Collected

Project	Effort	Number of Developers	Lines of Code (K)	Dev. Lines of Code (K)	Number of Components
SEL1	79.0	5	50.9	46.5	502
SEL2	39.6	4	75.4	31.1	490
SEL3	98.7	7	85.4	78.6	639

Table 3. Summary of Project Information

Project	Changes Per K Lines Of Developed Code	Errors Per K Lines Of Developed Code	Error To Mod Ratio (NonClericals Only)
SEL1	6.0	3.9	1.3
SEL2	7.4	3.8	.92
SEL3	9.7	3.9	.54

Table 4. Change and Error Densities

Project	Erroneous Change Rate (Ratio Of Changes Resulting In Errors To All Changes)	Errors Resulting From Change (As Percentage Of NonClericals)	Repeated Error Ratio (Average Number Of Corrections Per Error)
SEL1	.025	5	1.02
SEL2	.061	14	1.08*
SEL3	.041	12	1.05

* Upper bound. Exact number of repeated errors for SEL2 is unknown. by conservative means, the ratio could be estimated as 1.04.

Table 5. Measures of Erroneous Change

Project	Number Of People	Errors Per Person
SEL2	4	25
SEL1	5	26
SEL3	7	44

Table 6. Errors Per Person By Number Of People

Project	Effort (People-Months)	Errors Per Person-Month	Changes Per Person-Month
SEL2	39.6	2.4	.8
SEL1	79.0	1.7	3.6
SEL3	98.7	3.1	7.7

Table 7. Errors Per Effort By Effort

In the following sections each goal is satisfied by presenting conclusions based on the answers to the questions corresponding to the goal. Sections containing discussions of goals are headed by short descriptions of goals. Identifiers in parentheses following the goal descriptions are references to the goal, e.g. (G2) is a reference to goal 2. Not all goals are discussed here. Goal 5, "verify that concurrent data validation is needed," is discussed in a companion paper [1].

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. The latter explanation could be tested by analyzing the errors found in the projects during their use and maintenance. Attempting to satisfy this goal is beyond the scope of the research reported here.

Goal: Characterize Modifications (G1)

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 improving 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, these results suggest that most unplanned design modifications are small and only involve one component of the system. Several explanations are possible; either the programmers 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 modifications

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 distinguishes 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 nonlocalized with respect to individual components (most components that are changed are only changed once or twice), they are localized with respect to subsystem, i.e. the majority of changes are made in one or two subsystems.

Goal:
Characterize changes.

Questions:
What was the distribution of modifications according to the reason for the modification?

What was the distribution of changes across system components?

What was the distribution of effort required to design changes?

Goal:
Characterize errors.

Questions:
What was the distribution of errors according to the misunderstandings that caused them?

What was the distribution of effort required to correct errors?

What was the distribution of effort to correct errors across misunderstandings causing errors?

How many errors were the result of a software change?

Goal:
Characterize projects.

Goal:
Characterize programmers.

Goal:
Find factors that have significant effects on types and distributions of errors.

Goal:
Evaluate effectiveness of methodologies in NASA/GSFC environment.

Goal:
Suggest ways of improving NASA/GSFC software development practices.

Questions:
All questions are used in satisfying this goal. See list 1.

Goal:
Verify that concurrent data validation is needed.

Question:
How often must reported change data be corrected as a result of the data validation process?

List 3. Relationship Between Goals and Questions

Goal:

Identify good measures of correctness.

Questions:

What was the distribution of effort required to design changes?

What was the ratio of changes not made to correct an error to error corrections as a function of time during the development cycle?

What was the distribution of errors according to the misunderstandings that caused them?

What was the distribution of effort required to correct errors?

What was the distribution of effort to correct errors across misunderstandings causing errors?

How many errors were the result of a software change?

What was the distribution of errors across error detection techniques?

What was the number of attempted error corrections per error?

What was the ratio of errors to various measures often associated with effort and productivity?

What was the distribution of errors per person according to the number of people involved?

What was the number of errors for projects requiring memory overlays compared to those not requiring overlays?

What was the distribution of errors according to programmer?

Goals:

Identify effective techniques for detecting errors.

Question:

What was the distribution of errors across error detection techniques?

Goal:

Identify effective techniques for obtaining the information needed to correct errors.

Question:

What was the distribution of errors across error correction techniques?

Goal:

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?

Question:

What was the number of attempted error corrections per error?

List 3. Relationship Between Goals and Questions (continued)

Goal: Characterize Errors (G2)

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. The SEL developers apparently understand their requirements well enough that they can handle changes to them without much 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.

a. 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.

b. 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: Evaluate Effectiveness Of Methdologies In NASA/GSFC Environment (G3)

It was expected that various software engineering techniques would be tried in the course of these studies. However, it was found to be extremely difficult to characterize the different techniques and the differences in the ways in which the techniques were applied for the SEL projects reported here. Consequently, this goal could not be satisfied.

Goal: Suggest Ways Of Improving NASA/GSFC Software Development Practices (G4)

Previous analyses have shown that the most abundant source of errors lies in the process of designing and implementing individual components of the SEL projects. Improvements should come from the introduction of any techniques that assist the individual programmer in preventing and detecting errors. A number of techniques and tools have been suggested to help in this process. A few are listed in the following.

1. Program Design Language [9]
2. Code Reading and Inspections [10]
3. Program Proving [9, 11]
4. Programming By Stepwise Refinement [12]
5. Formal Specifications [13, 14]
6. Information Hiding [15]
7. Languages that provide strong typing, such as Pascal [16]

One would expect the introduction of some or all of these and other, similar techniques to perturb the SEL environment initially. After the initial learning period, if such techniques meet the claims made for them, a shift in the error distributions could be expected.

Goal: Identify Good Measures Of Correctness (G6)

In addition to various single parameters, one may also consider a number of different distributions as correctness measures. Candidates are the sources of nonclerical errors, the effort to design error corrections, the effort to isolate the error cause, the frequency distribution of error corrections, error corrections according to the subsystem in which they occur, and errors according to project phase.

Several of the preceding distributions serve to locate the most troublesome phases of the development process, and the most error-prone parts of the system. Others may be used as indicators of average difficulty in correcting errors.

Some conclusions concerning measures of correctness

It is not possible to identify from the data a single good parameter that can be used to measure correctness. Issues such as correctness relative to the amount of work that had to be done, or to the number of changes that had to be made, cannot easily be judged and cannot be discerned from a single parameter. Rather, a combination of parameters and distributions may be used to discover what and where difficulties were encountered in producing a particular system. Attempting to define the precise set of distributions and parameters to use is beyond the scope of this research. We do suggest that some of the following be used.

a. Ratio of errors to modifications, to give an indication of how the developers were spending their time;

b. Rate of erroneous changes, to give an indication of the difficulty the developers had in making changes;

c. Sources of changes and sources of errors, to give an indication of the kinds of problems the developers had to handle, and the kinds of difficulties they had;

d. Effort to make change, effort to isolate cause of error, and effort to design fix by source of error, to indicate difficulty of correcting errors;

e. Phase of entry of errors into the system, to indicate whether certain aspects of the development caused trouble, or whether difficulties tended to be spread out over the entire development.

Goal: Identify Effective Error Detection Techniques (G7)

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: Identify Effective Error Correction Techniques (G8)

It is clear from the data that the programmers favored code reading as an error correction technique. While this is not surprising, the lack of use of other techniques is surprising. Although we cannot determine if program reading is popular because programmers are writing programs that are easy to read, we can say that improving the readability of programs should improve the error correction process.

Goal: Investigate The Ripple Effect (G9)

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 (G10)

Examination of various parameters previously discussed 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 distribution. We can note variations in distributions that seem to distinguish some projects from others, and use the distinguishing distributions as the basis for more detailed experiments.

The proposed distinguishing distributions are listed in the following.

Change Distribution

The distribution of changes across modifications and nonclerical errors clearly distinguishes SEL3 from the other SEL projects.

Sources Of Modifications

The sources of modifications distributions 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 category. 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 distributions. SEL3 may be distinguished from the other SEL projects by its secondary peak in the "Design Multi-Comp" category. SEL1 shows a somewhat stronger peak in the "Fnl 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.

Effort To Isolate Error Cause

The distributions showing the effort to isolate error causes ap-

parently distinguish clearly between project SEL3 and the other SEL projects. (Because of the relatively large number of errors in the "Unknown" category in these distributions, the size of the distinction may not be as large as it appears.)

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 foregoing 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% 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 improving 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 requirements changes, these changes do not constitute a major source of errors. Interface errors are also not especially troublesome.

Goal: Characterize Programmers (G11)

Because there are few commonalities in the distributions of programmer errors, there is little that can be said to characterize the programmers as a group. Most have little trouble with the language or other attributes of the environment in which they program (e.g. the library system or the operating system). All of them seem to have the most problems in designing and implementing the internal structure of individual routines.

Goal: Find Factors That Significantly Affect Distributions Of Errors (12)

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 misunderstandings of specifications.

The error distributions for the SEL projects indicate that the single most important factor is the method used by the individual 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 previous designs in the development of new systems are in this category.

4. 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 shows that it was necessary to perform validation by means of developer interviews.) The data collected permit the following characterization 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 program and its documentation.

2. The most common type of error is one made in the design 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% of all changes take a day or less to correct. 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 individual components (most components that are changed are only changed once or twice), but localized with respect to subsystems (the majority of changes are

made in one or two subsystems).

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. Support facilities and techniques such as traces, dumps (which were once so popular that papers were published on how to read them e.g [17].), cross-reference and attribute listings, and program proving are rarely used.

Opportunities Missed

The data presented here and in [3, 2, 6] represent five years of data collection. During that time there was considerable and continuing consideration given to the appropriate 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 system. Programmers correcting their own errors, which was the usual case, could supply this data easily. One could then isolate errors that were not easily susceptible to detection by program execution or code reading. This example underscores the need for careful planning prior to the start of data collection.

Comparing Environments

In most sciences, valuable information is gained from repeating experiments, sometimes to confirm new results, other times to refine them. We believe this should be the case in Computer Science. Although some interesting patterns are exhibited in the SEL data, it would be useful to seek similar trends in data from environments. Unfortunately, there exists little comparable data ([4] is one exception). A primary reason 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 categorizations may be used to to ensure comparability.

Acknowledgments

The authors thank the many people at NASA/GSFC and Computer Sciences Corporation who filled out forms and submitted to interviews, especially Jean Grondalski and Dr. Gerald Page, and the librarians, especially Sam DePriest.

We thank Dr. John Gannon, Dr. Richard Meltzer, Frank McGarry, Dr. Gerald Page, Dr. David Parnas, Dr. John Shore, and Dr. Marvin Zelkowitz for their many helpful suggestions.

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

References

1. V. Basili and D. Weiss, "A Methodology For Collecting Valid Software Engineering Data," .
2. V. Basili and D. Weiss, "Evaluation of a Software Requirements Document By Analysis of Change Data," *Proc. Fifth Intntl. Conf. Software Engineering*, pp.314-323 (March 1981).
3. D. Weiss, "Evaluating Software Development by Error Analysis: The Data from the Architecture Research Facility," *J. Systems and Software* **1**, pp.57-70 (1979).
4. D. Weiss, "A Comparison of Software Errors In Different Environments," *NASA Software Enginerering Workshop* (November 1981).
5. V. Basili, M. Zelkowitz, F. McGarry, and others, "The Software Engineering Laboratory," Report TR-535, University of Maryland (May 1977).
6. S. Fryer and D. Weiss, "Evaluation of the A-7E Software Requirements Document By Analysis of Change Data: Two Years of Change Data," *15th Annual Asilomar Conference On Circuits, Systems, and Computers* (November 1981).
7. D. Weiss, "Evaluating Software Development By Analysis Of Change Data," TR-1120, University of Maryland Computer Science Center, College Park (November 1981).
8. J. Bailey and V. Basili, "A Meta-Model For Software Development Resource Expenditures," *Proc. Fifth Intntl. Conf. Software Engineering*, pp.107-116 (March 1981).
9. H. Mills, R. Linger, and B. Witt, *Structured Programming Theory and Practice*, Addison-Wesley, Reading (1979).
10. M. Fagan, "Design and Code Inspection and Process Control in the Development of Programs," TR 21.572, IBM System Development Division (December 1974).
11. E. W. Dijkstra, *A Discipline of Programming*, Prentice-Hall, Englewood Cliffs (1976).
12. N. Wirth, "Program Development by Stepwise Refinement," *Comm. ACM* **14**(4), pp.221-227 (April 1971).
13. D. L. Parnas, "A Technique For Software Module Specification With Examples," *Comm. ACM* **15**(5), pp.330-336 (May 1972).
14. J. Guttag, "The Specification and Application to Programming of Abstract Data Types," CSRG-59, University of Toronto Dept. of Computer Science Computer Systems Research Group (1975).

15. D. L. Parnas, "On the criteria to be used in decomposing systems into modules," *Comm. ACM* **15**(12), pp.1053-1058 (December 1972).

16. K. Jensen and N. Wirth, *Pascal User Manual and Report Second Edition*, Springer-Verlag, New York (1974).

17. D. Norris, "An Introduction To OS/360 MVT Control Logic And Debugging With MVT Core Dumps," *IBM Technical Information Exchange* (January 1969).

Appendix

Answering Questions of Interest

The questions of interest are answered by presenting and analyzing the data distribution(s) associated with each question. For each question there is a short discussion of the associated 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 8 shows the relation between the questions and the distributions. Not all questions are discussed here. Question 17, "How often must reported change data be corrected as a result of the data validation process?" is discussed in a companion paper [1].

For some questions either there were insufficient data to answer the questions, or the data were judged insufficiently reliable to produce meaningful distributions. Interpretations of the questions as they relate to the goals of the studies are given in a later section.

One purpose of this research is to provide a set of empirically-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 the data categorizations previously discussed. The following sections are intended to help the reader understand the organization and content of the various histograms and tables.

Organization of Data Presentation

In general, the histograms are organized into figures, with each figure containing corresponding histograms for all projects. Examples are figure 1, which shows a broad view of all change data, and figure 3, which shows the sources of nonclerical errors for all projects. For some figures, not all projects are represented, since a particular set of data may not be relevant or available for some projects.

Tables are used to show the relationship between two different categorizations, such as effort to design modification according to source of modification (table 9). Labels on the histograms and tables are generally mnemonic abbreviations of descriptions of data categories (e.g. PE means planned enhancement). Keys, supplied for non-obvious labels, provide the complete name for each mnemonic.

Data Categorization

During the data collection period, several improvements were made to the forms. One result is that forms for some 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 projects within a particular figure. Such differences in categorization 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 figure 10, "Effort To Change NonClerical 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.

1.	What was the distribution of modifications according to the reason for the modification?	Figures 3, 4
2.	What was the distribution of changes across system components?	Figures 14, 15
3.	What was the distribution of effort required to design changes?	Figures 8, 9, 10
4.	What was the ratio of changes not made to correct an error to error corrections as a function of time during the development cycle?	Data not sufficiently reliable to produce meaningful distribution.
5.	What was the distribution of errors according to the misunderstandings that caused them?	Figures 5, 6, 7
6.	What was the distribution of effort required to correct errors?	Figures 10, 11, 12, 13
7.	What was the distribution of effort to correct errors across misunderstandings causing errors?	Tables 11, 12, 13, 14, 15, 16
8.	How many errors were the result of software changes?	Table 5
9.	What was the distribution of errors across error detection techniques?	Tables 17, 18, 19
10.	What was the distribution of errors across error correction techniques?	Tables 20, 21, 22
11.	What was the number of attempted error corrections per error?	Table 5
12.	What was the distribution of error corrections across project phases?	Figure 18
13.	What was the ratio of errors to various measures often associated with effort and productivity?	Tables 4, 5, 6, 7
14.	What was the distribution of errors per person according to the number of people involved?	Table 6
15.	What was the number of errors for projects requiring memory overlays compared to those not requiring overlays?	Insufficient data for meaningful results.
16.	What was the distribution of errors according to programmer?	Figure 19
17.	How often must reported change data be corrected as a result of the data validation process?	Presented elsewhere

Table 8. Figures/Tables used in Answering Questions

Organization of Data Presentation

In general, the histograms are organized into figures, with each figure containing corresponding histograms for all projects. Examples are figure 1, which shows a broad view of all change data, and figure 3, which shows the sources of nonclerical errors for all projects. For some figures, not all projects are represented, since a particular set of data may not be relevant or available for some projects.

Tables are used to show the relationship between two different categorizations, such as effort to design modification according to source of modification (table 9). Labels on the histograms and tables are generally mnemonic abbreviations of descriptions of data categories (e.g. PE means planned enhancement). Keys, supplied for non-obvious labels, provide the complete name for each mnemonic.

Data Categorization

During the data collection period, several improvements were made to the forms. One result is that forms for some 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 projects within a particular figure. Such differences in categorization 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 figure 10, "Effort To Change NonClerical 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 reported in this category that it was important to subcategorize further. (The next version of the form contained the new subcategories explicitly). Figure 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 2 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 figure 10) 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 figure 5, values in the distributions are shown as percentages of nonclerical 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 .5% of the distribution are shown as containing 1%. (Categories that contain no data do not appear in the distributions.) For no distribution does this make a difference of more than 1% in any category. For some distributions, there is a resulting round-off error.

Answers To The Questions

In the following sections we discuss the answers to the questions of interest. For some questions, the data are not sufficiently complete or accurate to provide meaningful or reliable answers. The reasons for this have been discussed in previous sections; where necessary, they are elaborated. Sections are headed by short descriptions of questions. Identifiers in parentheses following the question descriptions are references to the question number, e.g. (Q2) is a reference to question 2.

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.

Figures 1 and 2 show some interesting differences among the three projects. The proportion of both all errors and of nonclerical errors declines from SEL1 (64% and 47% respectively) through SEL3 (40% and 32% 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? (Q1)

Modification distributions are shown in figure 3. 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.

Similarities 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.

Figure 4 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? (Q2)

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, figure 14 shows the percentage of components that were changed once, twice, etc. As an example, for SEL1, 29% of the components were changed once, and 30% were changed twice.

The frequency distributions for all the SEL projects show the same pattern: 50% or more of the components that were changed were only changed once or twice, and more than 90% were changed 6 times or less. The pattern is even more pronounced for fixes (figure 15): 70% or more of the fixed components were only fixed once or twice.

Figure 16 shows the patterns of subsystems that are changed and fixed most often. (The distributions are obtained by grouping the data for the components into subsystems.) It is clear from these distributions that at most 2 or 3 of the subsystems receive the most attention.

What was the distribution of effort required to design changes? (Q3)

Change effort distributions are shown in figures 8 through 13. Examining figure 8, which shows the effort for all changes except clerical errors, one can see that most (more than 75% of) changes fall into the easy or medium categories for all SEL projects. Figure 9, which is restricted to modifications only, shows a similar, but not as strong, trend. The trend is most pronounced for nonclerical errors.

What was the distribution of errors according to the misunderstandings that caused them? (Q5)

Inspection of the distributions showing sources of nonclerical errors (figure 5) 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 specifications are minor sources of errors. (Note that figure 3 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 (figure 7).

Further analysis of the errors committed in design and implementation of components is shown in figure 6. In the SEL environment, data errors (errors in the value or structure of data) are either about evenly balanced with or predominate errors in the intended use of compnenets.

What was the distribution of effort required to correct errors? (Q6)

Effort distributions for correcting errors are shown in figure 10. (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% or more of the errors were corrected in one hour or less, and more than 85% were corrected in one day or less.

As might be expected, the distributions for effort expended in finding error causes (figures 11, 12, and 13) follow a similar pattern. From these results we may conclude that the programmers tend to spend their time finding and correcting many "small" errors rather than few "large" errors.

What was the distribution of effort to correct errors across misunderstandings causing errors? (Q7)

Tables 11 through 16 support the view of most errors as being easy to find and fix and as occurring in component design or implementation. Very few errors take more than a day of effort to fix. Although interface errors are often cited as being particularly difficult to correct, table 13 shows that they follow the same pattern as other subcategories of errors.

The only deviation from the pattern appears to occur in the effort to fix requirements and specification errors, where the distribution between easy and medium rated errors is more balanced than for the other subcategories. These results suggest that requirements and specification errors are more difficult to correct than others. However, the small number of errors in these subcategories makes it dangerous to draw such a conclusion.

How many errors were the result of a software change? (Q8)

Table 5 shows that the SEL projects handled changes with little trouble; relatively few errors were the result of a change to the software.

What was the distribution of errors across error detection techniques? (Q9)

The relative frequency of use of various error detection techniques are shown in tables 17 through 19 for the SEL projects. While examining the distributions, one must recall that SEL change monitoring did not begin until code was baselined 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 distribution of errors across error correction techniques? (Q10)

The relative frequency of use of various error correction techniques are shown in tables 20 through 22. While it is not surprising that code reading by the programmer dominates all other methods, the relative infrequency of techniques such as traces, special debug code, test runs, and reading documentation is somewhat surprising. Dumps, which were once so popular that papers were published on how to read them (e.g. [17]), were rarely used.

What was the number of attempted error corrections per error? (Q11)

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 5, both of these effects appear quite small. The worst case is about 6% of the changes resulting in errors (SEL2). The errors resulting from change for the worst case (SEL2) comprised 14% 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).

What was the distribution of error corrections across project phases? (Q12)

The distributions of errors according to the phase of the project in which the error entered the system are shown in figure 18. All projects show a strong spike in the code and test phase. These distributions are somewhat less reliable than others because programmers could not always decide exactly when a particular error occurred. The unknown subcategory comprises such cases.

What was the ratio of errors to various measures often associated with effort and productivity? (Q13)

What was the distribution of errors per person according to the number of people involved? (Q14)

Because of their similarity, questions 13 and 14 are answered together.

Tables 4 through 7 show a variety of ways of normalizing error rates to productivity measures. Each normalization may be used to rank the projects. For the six different normalizations there are six different rankings.

What was the distribution of errors according to programmer? (Q16)

Distributions of errors for individual programmers are shown in figure 9. As with the project error distributions (e.g. figure 5), the individual programmer

error distributions all show peaks in the "Design Single Comp" category. Both the relative size of this peak and the variation over the remainder of the distribution is considerably more variable among the different programmers than among the different projects.

	Easy LE 1 HR	Medium 1Hr To 1 Day	Hard GT 1 Day	Unknown
Req	1		2	
Design	33	22	6	1
Debug	8	2		
Env	1	1		
PE	11	5	3	1
Other		3		

SEL1

	Easy LE 1 HR	Medium 1 Hr To 1 Day	Hard GT 1 Day	Unknown
Req	11	6	9	4
Design	21	19	8	4
Debug	3	1		
Env	4			
PE	4	3	4	1
Unknown	2			

SEL2

	Easy LE 1 HR	Medium 1 Hr To 1 Day	Hard 1 Day to 3 Days	Formidable GT 3 Days	Unknown
Req	6	10	3	5	1
Design	34	9	2	1	.5
Debug	3	2	.5		1
Env	.5				
PE	7	9	5	4	.5
Unknown		.5		.5	

SEL3

Table 9. Effort To Modify By Source of Mod
(As Percentage of Total Mods)

Key

Design	Modifications caused by changes in design
Debug	Modifications to insert or delete debug code
Env	Modifications caused by changes in the hardware or software environment
PE	Planned Enhancements
Req	Modifications caused by changes in requirements of functional specifications
Unknown	Causes of these modifications are not known

	Easy LE 1 HR	Medium 1 Hr to 1 Day	Hard GT 1 Day	Unknown
Clarity	2	3	3	
US	12	7	1	1
Opt	15	11	2	
Unknown	4	1		

SEL1

	Easy LE 1 HR	Medium 1 Hr to 1 Day	Hard GT 1 Day	Unknown
Clarity	6	4	1	
US	5	5		
Opt	7	4	4	1
Other		1		
Unknown	3	5	3	

SEL2

	Easy LE 1 HR	Medium 1 Hr to 1 Day	Hard 1 Day to 3 Days	Formidable GT 3 Days	Unknown
Clarity	28	3	1	1	
US	3.5	5	.5	1	
Opt	2	2	.5		

SEL3

Table 10. Effort to Modify By Source of Mod (Design Mods Only)
(As Percentage of Total Mods)

Key

Clarity	Improvement of clarity, maintainability, or documentation
Opt	Optimization of time/space/accuracy
Unknown	Causes of these design changes are not known
US	Improvement of user services

	Easy LE 1 HR	Medium 1 Hr To 1 Day	Hard GT 1 Day	Unknown
Req	1	1		
Fnl Spec	8	4	2	
Design	5	2		1
Multi-Comp Design/Impl	45	18	2	1
Single Comp Lang/Compiler	1			
Env		2		
Other	5	2		1

SEL1

	Easy LE 1 HR	Medium 1 Hr To 1 Day	Hard GT 1 Day	Unknown
Req	2	2		
Fnl Spec	1			2
Design	2	1	1	
Multi-Comp Design/Impl	41	26	2	9
Single Comp Lang/Compiler	7	1		1
Env		1		
Other	2	1		

SEL2

	Easy LE 1 HR	Medium 1 Hr To 1 Day	Hard 1 Day To 3 Days	Formidable GT 3 Days	Unknown
Req	2	3	1		
Fnl Spec	2	3	1		
Design	9	12	2	1	1
Multi-Comp Design/Impl	32	20	2		2
Single Comp Lang/Compiler	1	2			
Env	2	1			1
Other	1				

SEL3

Table 11. Effort To Design Fix By Source Of Error
(As Percentage of NonClerical Errors)

Key

Design Multi-comp	Design error involving several components
Design/Impl Single Comp	Error in the design or implementation of a single component
Env	Misunderstanding of external environment, except language
Fnl Spec	Functional specifications incorrect or misinterpreted
Lang	Error in use of programming language/compiler
Req	Requirements incorrect or misinterpreted

	Easy LE 1 HR	Medium 1 Hr To 1 Day	Hard GT 1 Day	Unknown
Intended Use	20	16	2	1
Data	29	5	1	1
Other	1			

SEL1

	Easy LE 1 HR	Medium 1 Hr To 1 Day	Hard GT 1 Day	Unknown
Intended Use	16	11	3	7
Data	28	16		2

SEL2

	Easy LE 1 HR	Medium 1 Hr To 1 Day	Hard 1 Day To 3 Days	Formidable GT 3 Days
Intended Use	12	13	2	
Data	29	18	2	1

SEL3

Table 12. Effort To Design Fix By Source Of Error (Design Errors Only)
(As Percentage Of NonClerical Errors)

Key

Data	Error in the use of data
Intended Use	Error in intended function, i.e. program behavior does not correspond to the intended use of the program

Project	Easy LE 1 HR	Medium 1 Hr To 1 Day	Hard GT 1 Day	Formidable	Unknown
SEL1	8	4			1
SEL2	5	2	2		
SEL3	11	13	2	1	

Table 13. Effort To Design Fix For Interface Errors
(As Percent Of NonClerical Errors)

	Easy LE 1 HR	Medium 1 Hr To 1 Day	Hard GT 1 Day	NA	Unknown
Req	1	1			
Fnl Spec	2	4		5	3
Design Multi-Comp	2	3			2
Design/Impl Single Comp	31	26	2	2	5
Lang/Compiler	1				
Env				1	1
Other	1			7	

SEL1

	Easy LE 1 HR	Medium 1 Hr To 1 Day	Hard GT 1 Day	NA	Unknown
Req	3			1	
Fnl Spec		1		1	1
Design Multi-Comp	1	2	1		
Design/Impl Single Comp	27	32	1	4	12
Lang/Compiler	3	2	1	1	2
Env	1				
Other				1	2

SEL2

	Easy LE 1 HR	Medium 1 Hr To 1 Day	Hard GT 1 Day	NA	Unknown
Req	2	3	1		1
Fnl Spec	2	2			1
Design Multi-Comp	13	8	1		4
Design/Impl Single Comp	35	17	1		4
Lang/Compiler	2	2			
Env	1	1			1
Other	1				

SEL3

Table 14. Effort To Isolate Cause By Source Of Error
(As Percentage Of NonClerical Errors)

Key

Design Multi-Comp	Design error involving several components
Design/Impl Single Comp	Error in the design or implementation of a single component
Env	Misunderstanding of external environment, except language
Fnl Spec	Functional specifications incorrect or misinterpreted
Lang	Error in use of programming language/compiler
Req	Requirements incorrect or misinterpreted

	Easy LE 1 HR	Medium 1 Hr To 1 Day	Hard GT 1 Day	NA	Unknown
Intended Use	17	17			
Data	16	12	2	2	3
Other				1	

SEL1

	Easy LE 1 HR	Medium 1 Hr To 1 Day	Hard GT 1 Day	NA	Unknown
Intended Use	9	13	1	4	10
Data	19	21	1		2
Other					

SEL2

	Easy LE 1 HR	Medium 1 Hr To 1 Day	Hard GT 1 Day	NA	Unknown
Intended Use	16	11	1		1
Data	32	13	2		5

SEL3

Table 15. Effort To Isolate Cause By Source Of Error (Design Errors Only) (As Percentage Of NonClerical Errors)

Key

Data	Error in the use of data
Intended Use	Error in intended function, i.e. program behavior does not correspond to the intended use of the program

Project	Easy LE 1 HR	Medium 1 Hr To 1 Day	Hard GT 1 Day	NA	Unknown
SEL1	5	4		1	3
SEL2	3	4	1		1
SEL3	14	9	1	2	

Table 16. Effort To Isolate Cause For Interface Errors
(As Percent Of NonClerical Errors)

	Activities Used For Detection	Error First Detected By
Test Runs	128	93
Code Reading By Programmer	59	40
Code Reading By Other Person	21	15
Reading Documentation	1	1
Proof Technique		
Trace		
Dump	1	
Cross Reference	5	1
Attribute List	1	
Special Debug Code	11	3
General Error Messages	3	1
Project Specific Error Messages		
Inspection Of Output	12	7
Other	4	7

Table 17. SEL1 Frequency Of Use Of Error Detection Techniques

	Activities Used For Detection	Error First Detected By
Test Runs	83	46
Code Reading By Programmer	73	18
Code Reading By Other Person	56	22
Reading Documentation	4	
Proof Technique		
Trace	4	
Dump	5	1
Cross Reference	1	
Attribute List	2	1
Special Debug Code	4	
General Error Messages	12	5
Project Specific Error Messages	2	1
Inspection Of Output	46	33
Other		

Table 18. SEL2 Frequency Of Use Of Error Detection Techniques

	Activities Used For Program Validation	Activities Successful In Detecting Error Symptoms
Pre-acceptance Test Runs	162	96
Acceptance Testing	27	21
Post Acceptance Use	9	8
Inspection Of Output	143	129
Code Reading By Programmer	188	88
Code Reading By Other Person	115	17
Talks With Other Programmers	7	9
Special Debug Code	12	3
System Error Messages	15	13
Project Specific Error Messages	5	5
Reading Documentation	3	2
Trace		
Dump	4	4
Cross Reference Or Attribute List	6	6
Proof Technique		
Other	4	4

Table 19. SEL3 Frequency Of Use Of Error Detection Techniques

	Activities Tried To Isolate Cause	Activties Successful In Isolating Cause
Test Runs	13	6
Code Reading by Programmer	134	129
Code Reading by Other Person	24	22
Reading Documentation		
Proof Technique		
Trace		
Dump		
Cross Reference	3	3
Attribute List		
Special Debug Code	4	2
General Error Messages		
Project Specific Error Messages	1	
Inspection Of Output	9	1
Other	1	1

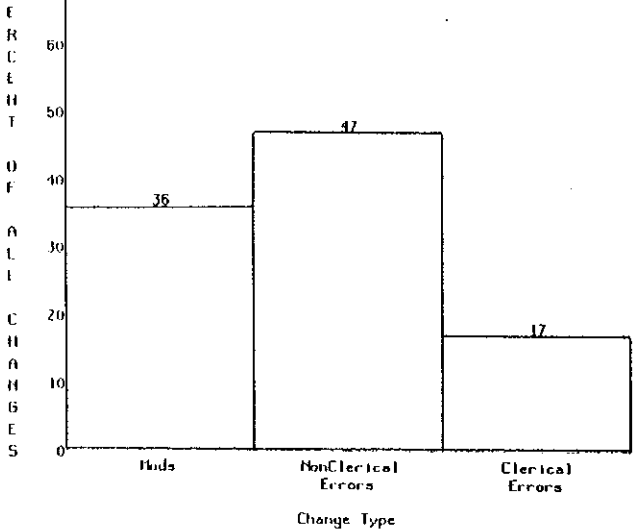
Table 20. SEL1 Frequency Of Use Of Error Correction Techniques

	Activities Tried To Isolate Cause	Activities Successful In Isolating Cause
Test Runs	9	5
Code Reading By Programmer	71	69
Code Reading By Other Person	38	34
Reading Documentation	3	
Proof Technique		
Trace		
Dump	5	2
Cross Reference		
Attribute List	1	1
Special Debug Code	1	4
General Error Messages	1	2
Project Specific Error Messages	1	
Inspection Of Output	11	8
Other		

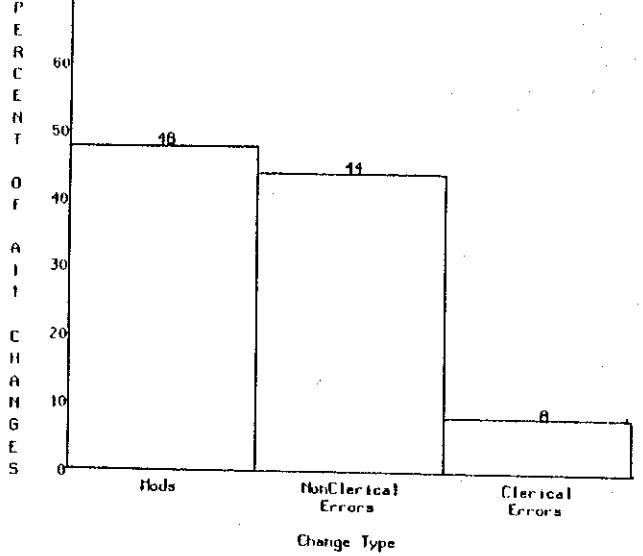
Table 21. SEL2 Frequency Of Use Of Error Correction Techniques

	Activities Tried To Find Cause	Activities Successful In Finding Cause
Pre-acceptance Test Runs	7	4
Acceptance Testing		
Post Acceptance Use		
Inspection Of Output	65	39
Code Reading By Programmer	224	220
Code Reading By Other Person	42	38
Talks With Other Programmers	23	20
Special Debug Code	5	3
System Error Messages		
Project Specific Error Messages	3	1
Reading Documentation	13	9
Trace	1	1
Dump	3	3
Cross Reference Or Attribute List	2	
Proof Technique		
Other		

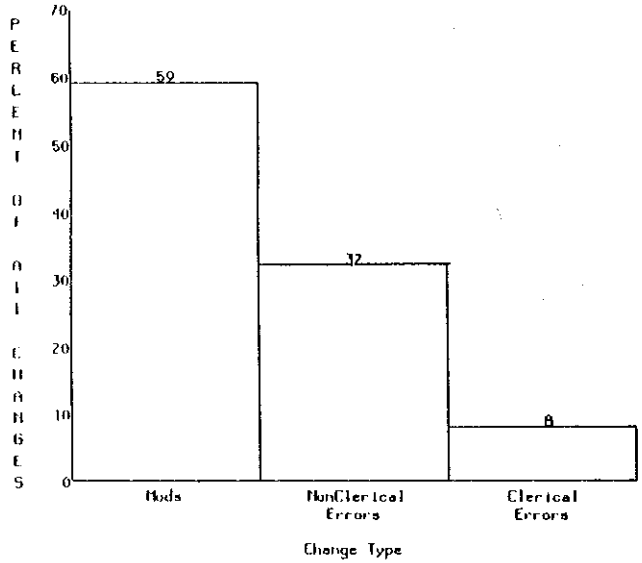
Table 22. SEL3 Frequency Of Use Of Error Correction Techniques



SEL1

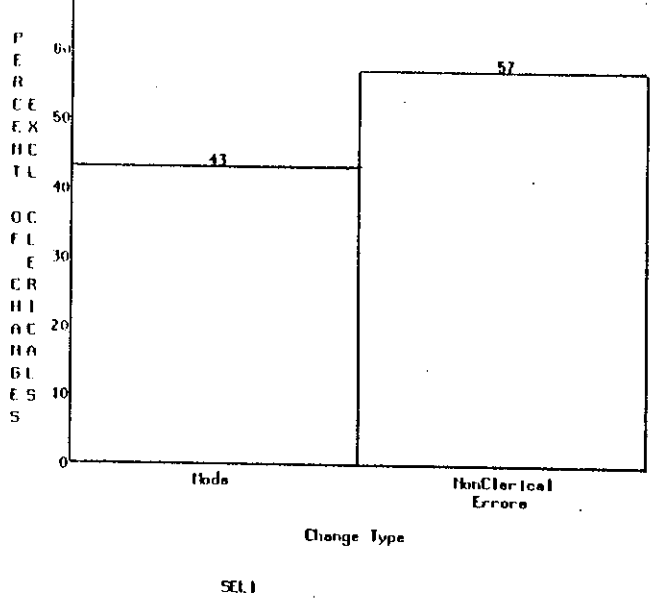


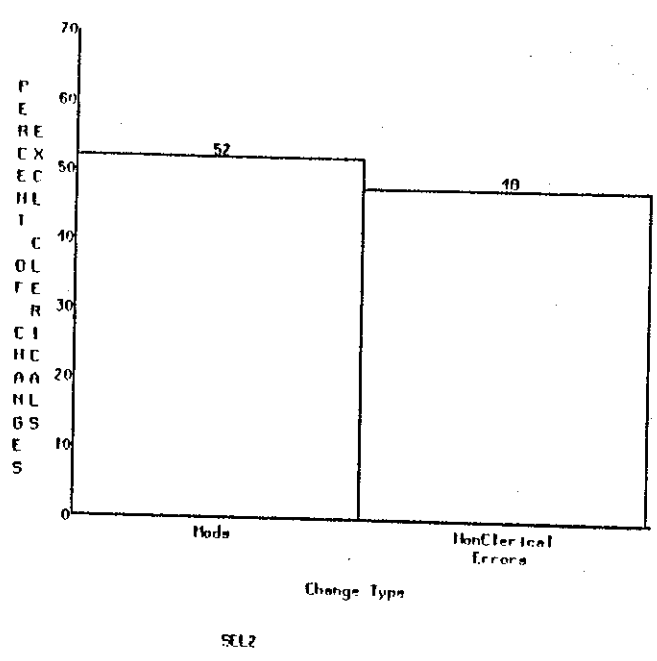
SEL2



SEL3

FIGURE 1 CHANGES





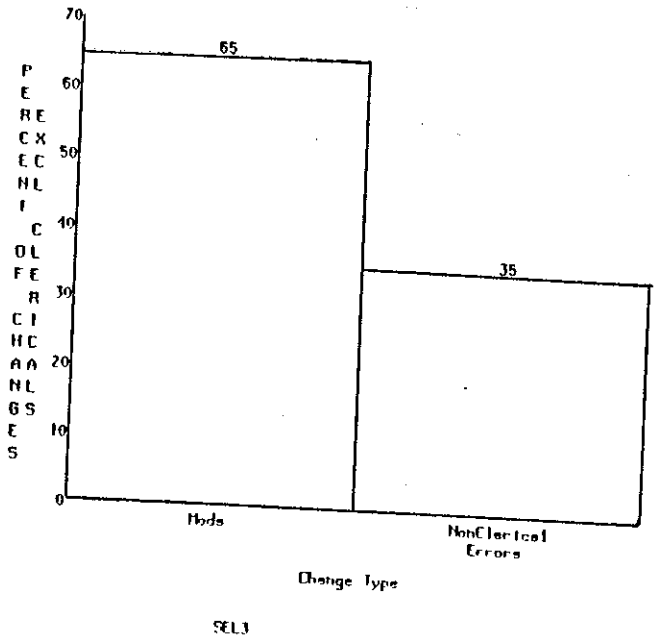
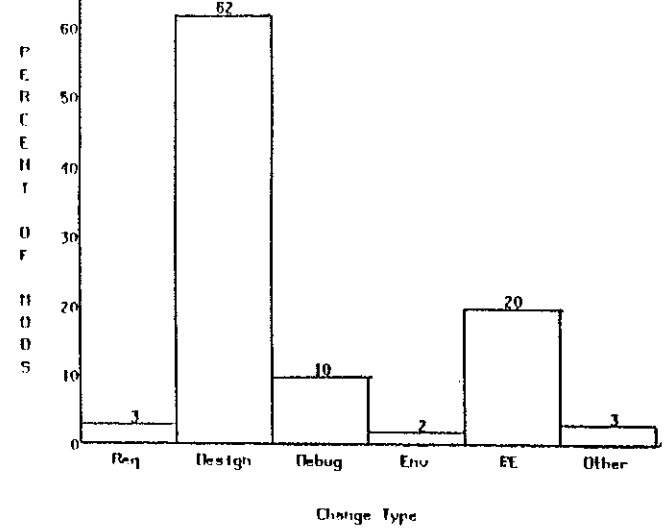
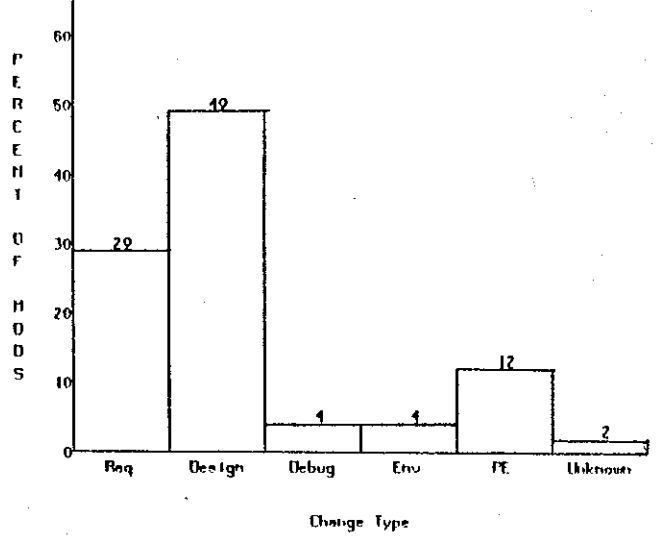


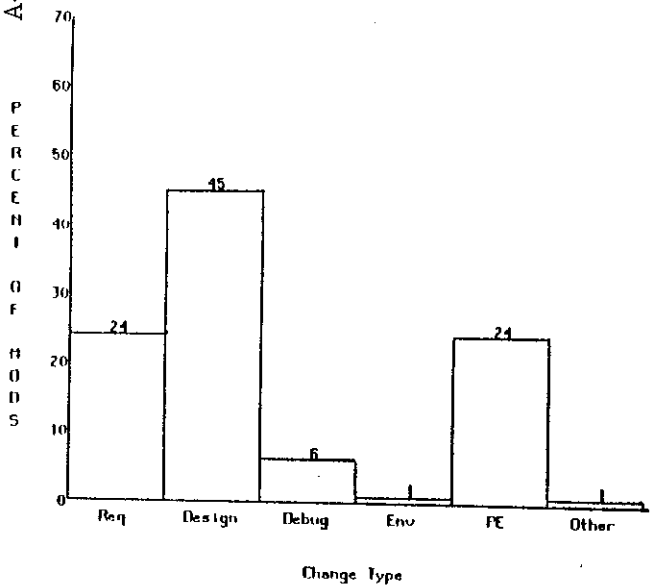
FIGURE 2 CHANGES (CLERICAL ERRORS EXCLUDED)



SEL1

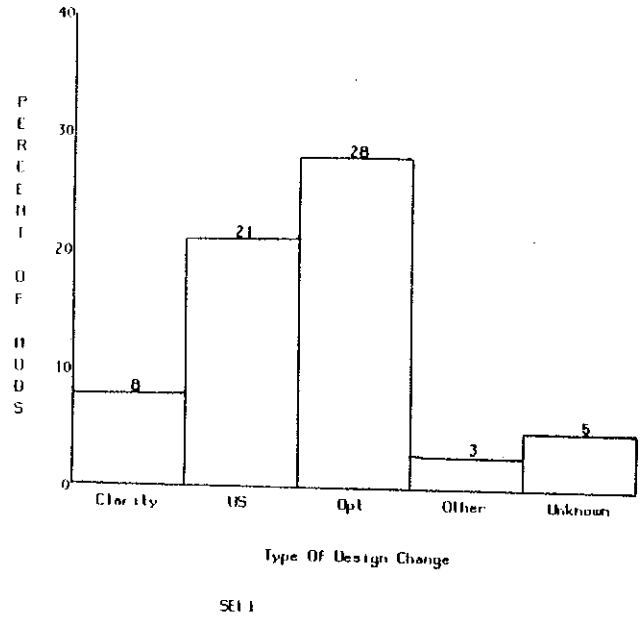


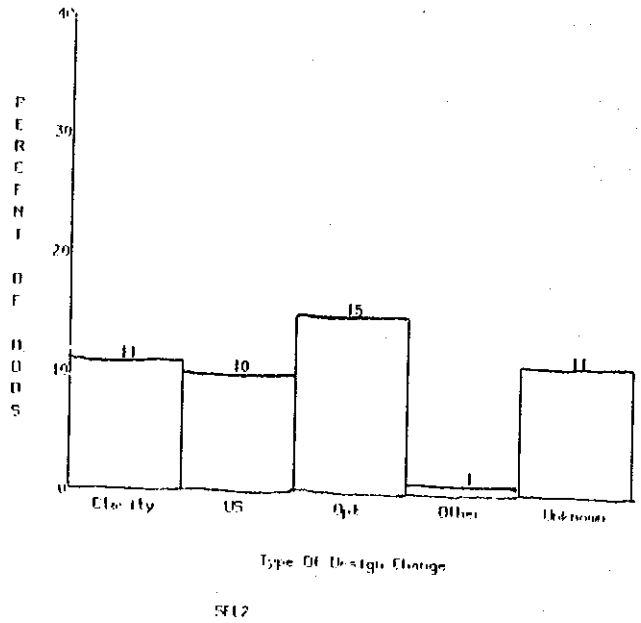
SEL2



SEL3

FIGURE 3 SOURCES OF MODIFICATIONS





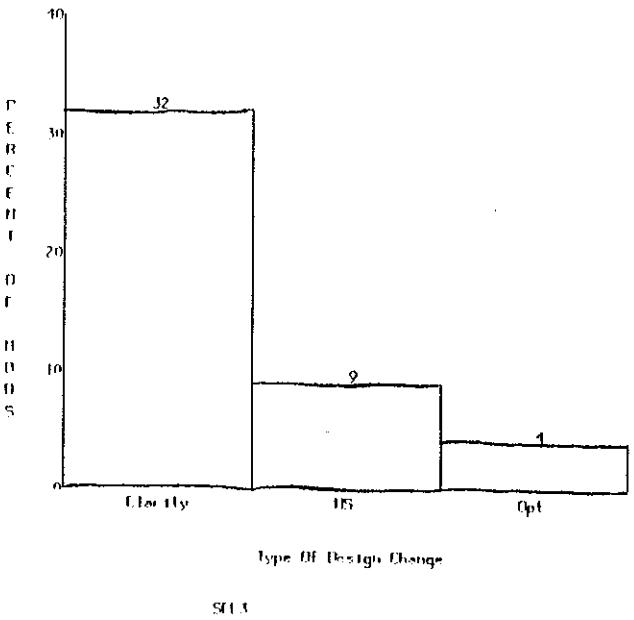
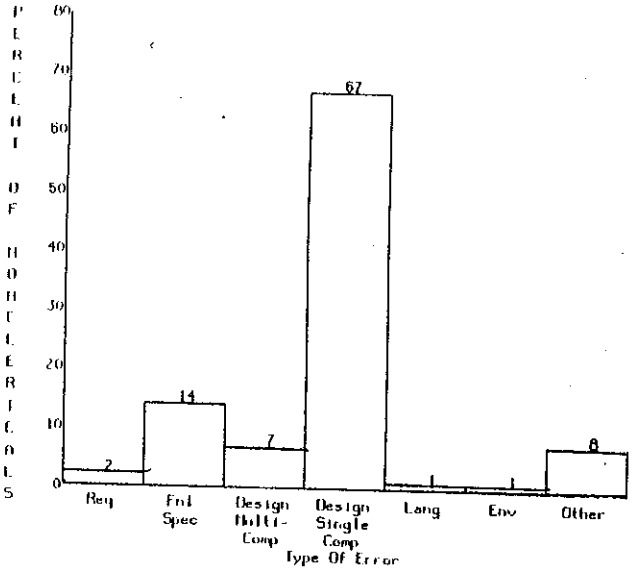
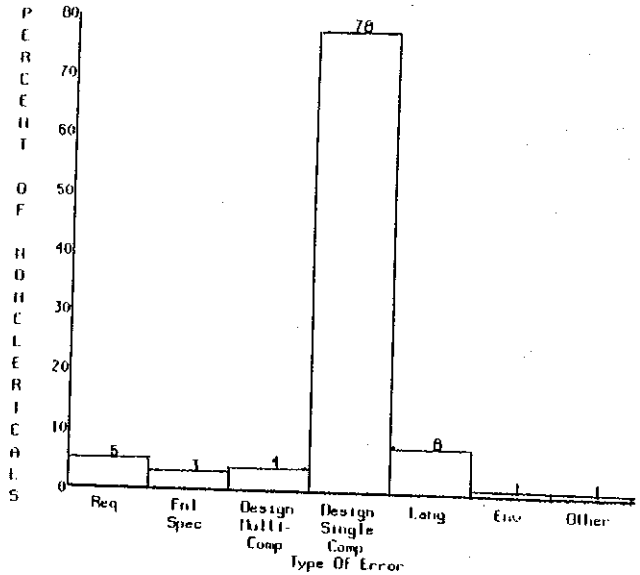


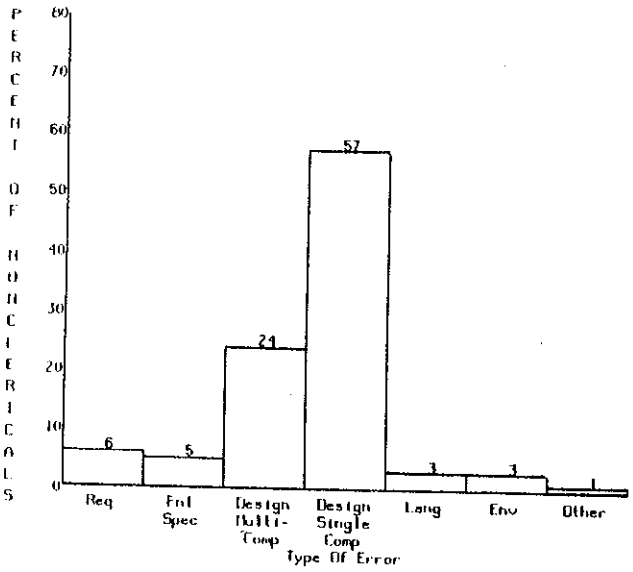
FIGURE 4 SOURCES OF DESIGN MODS



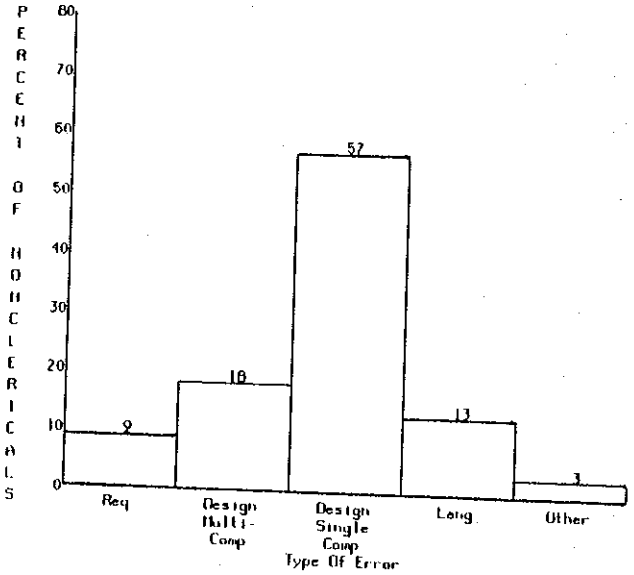
SEL1



SEL2

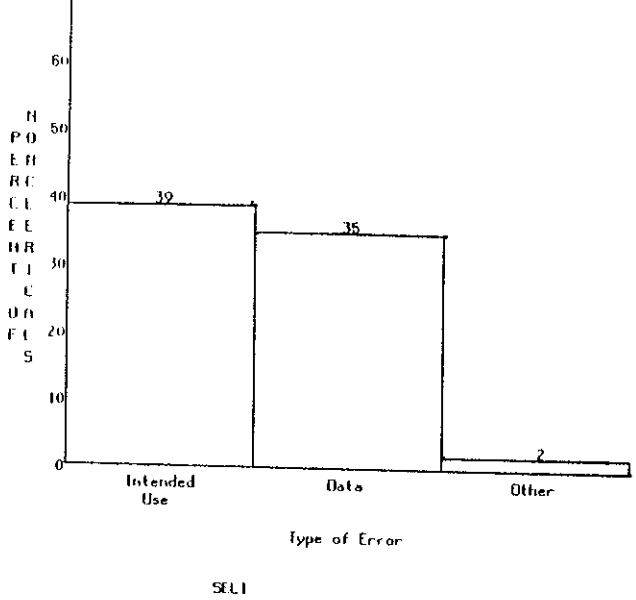


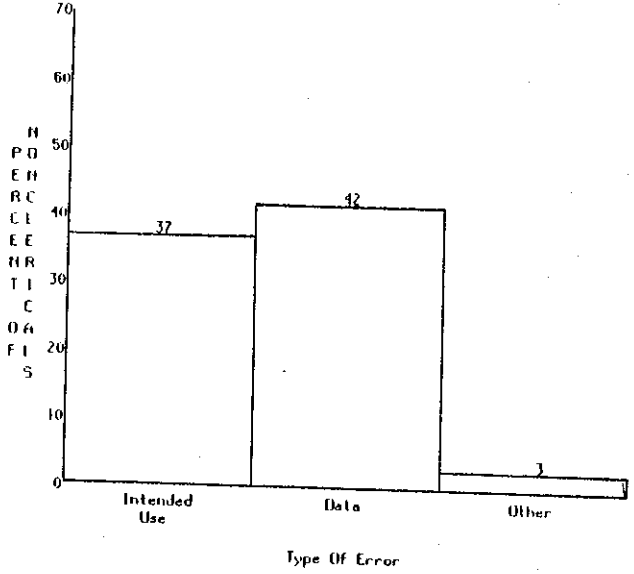
SEL3

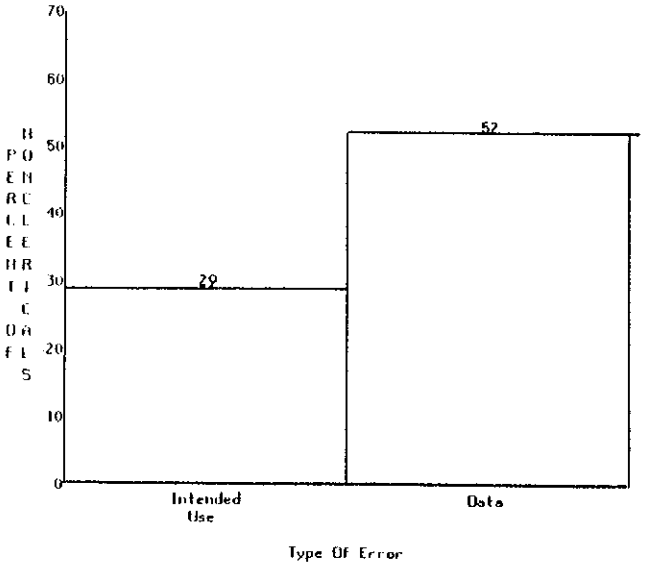


ARF

FIGURE 5 SOURCES OF NONCLERICAL ERRORS







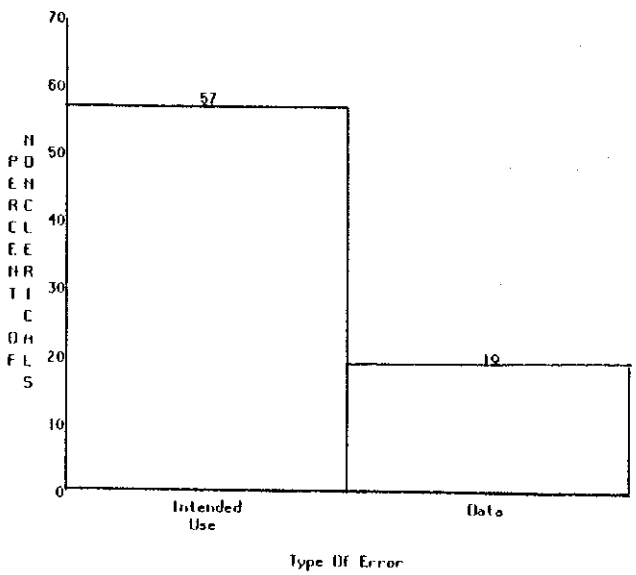


FIGURE 6 SOURCES OF DESIGN/IMPLEMENTATION ERRORS

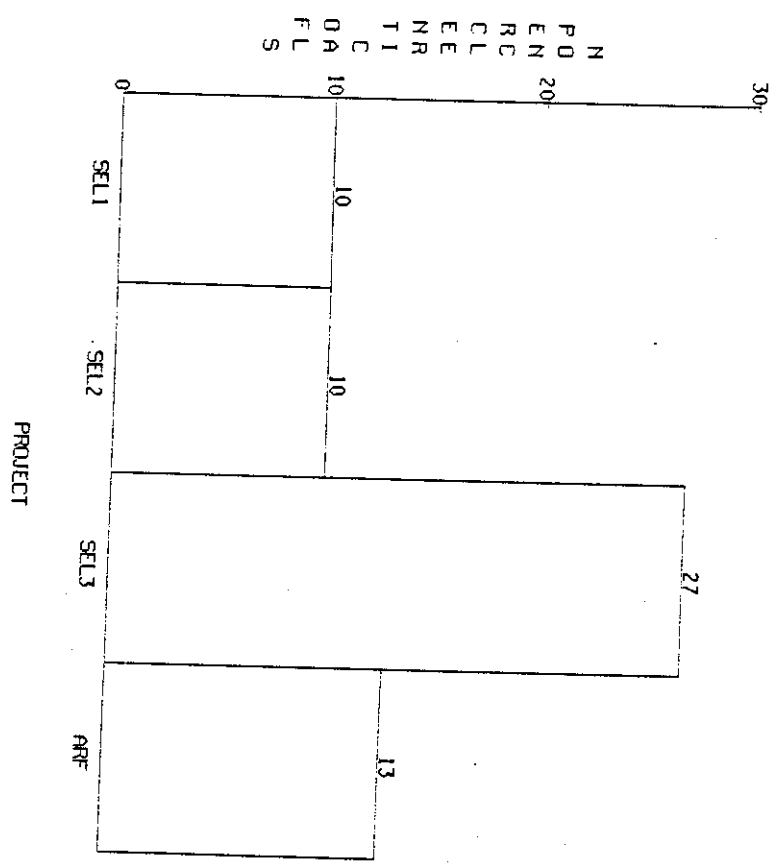
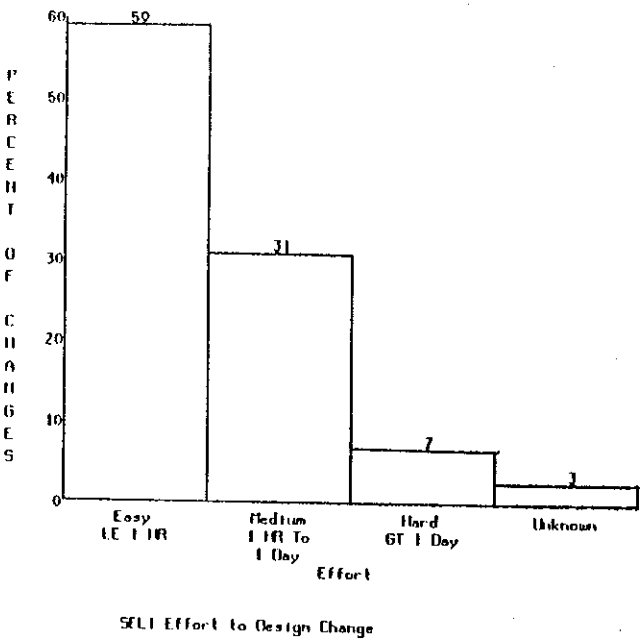
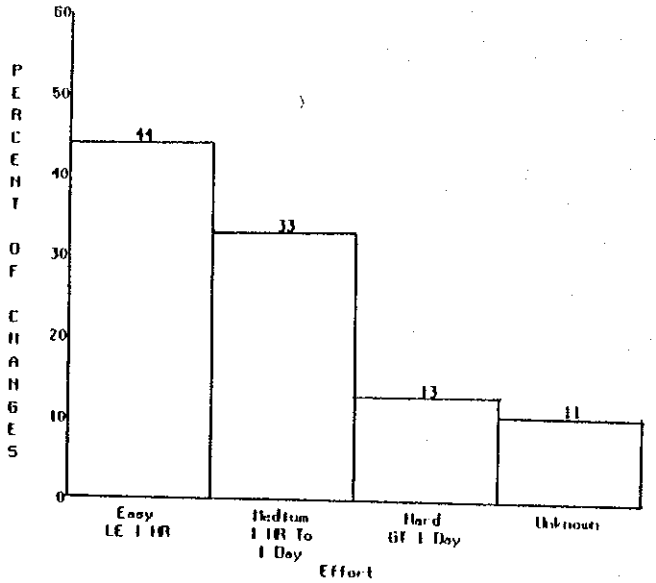


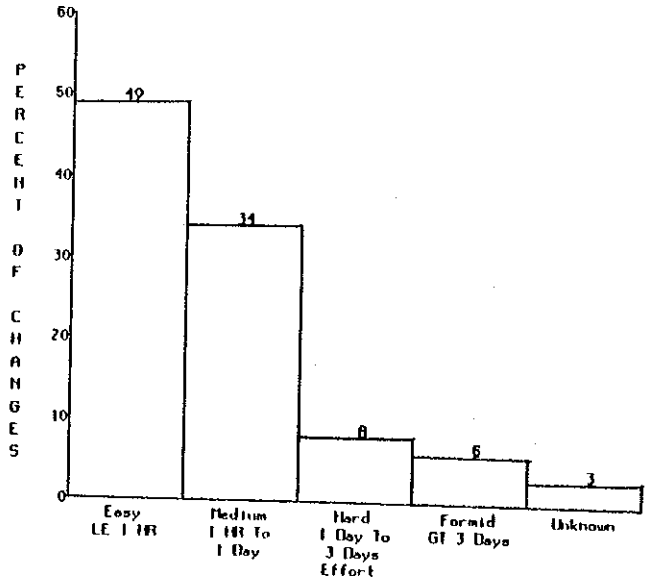
FIGURE 7 INTERFACE ERRORS



SEL1 Effort to Design Change

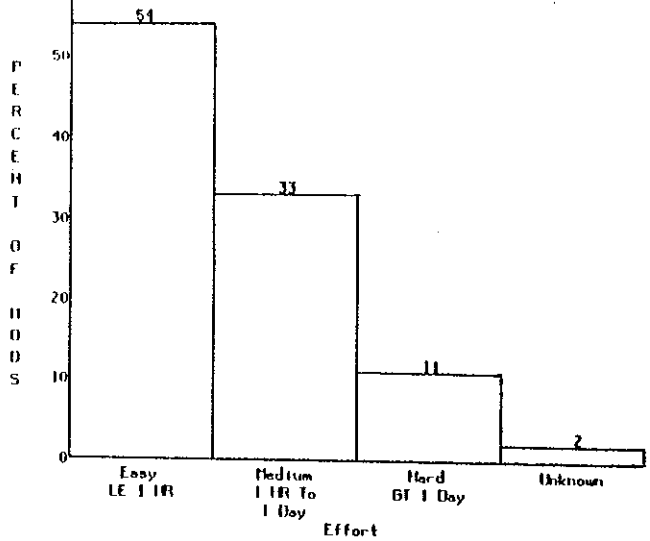


SEL2 Effort to Design Change

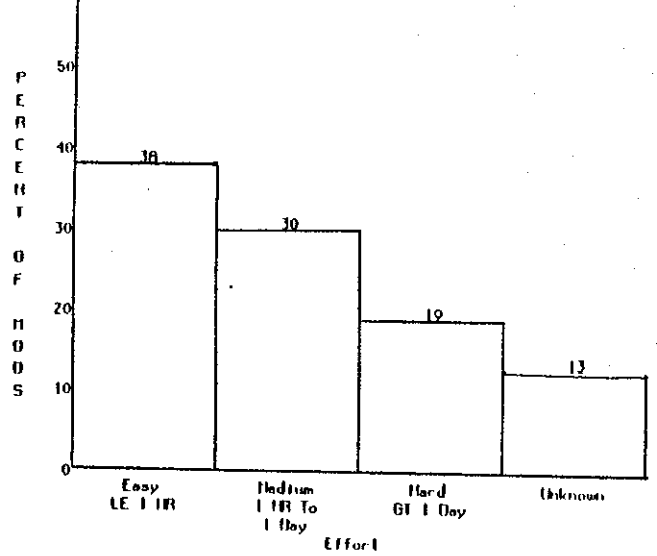


SEL3 Effort to Make Change

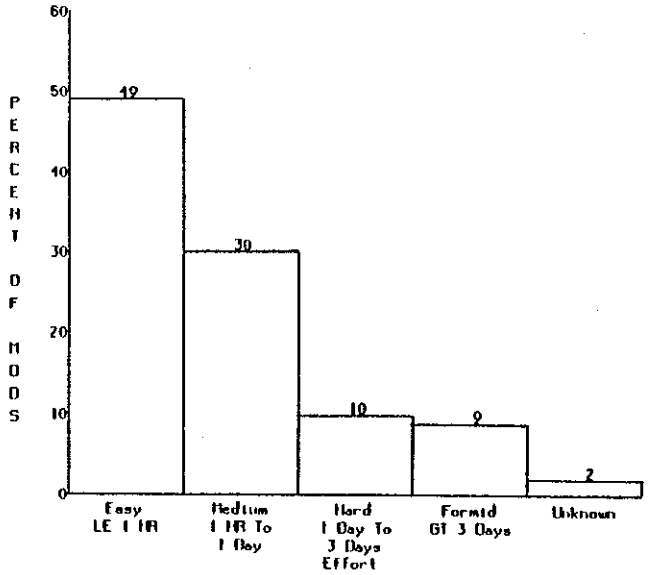
FIGURE 8 EFFORT TO CHANGE
MODIFICATIONS AND CLERICAL ERRORS



SEL1 Effort to Design Change

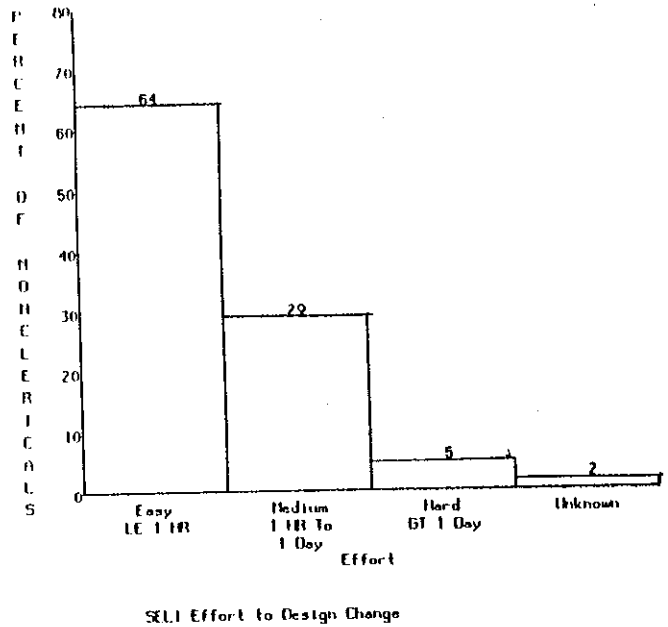


SEL2 Effort to Design Change

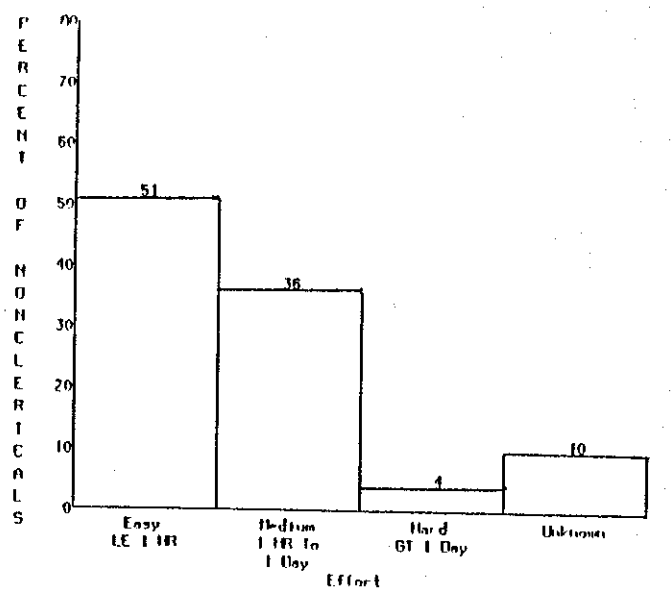


SEL3 Effort to Make Change

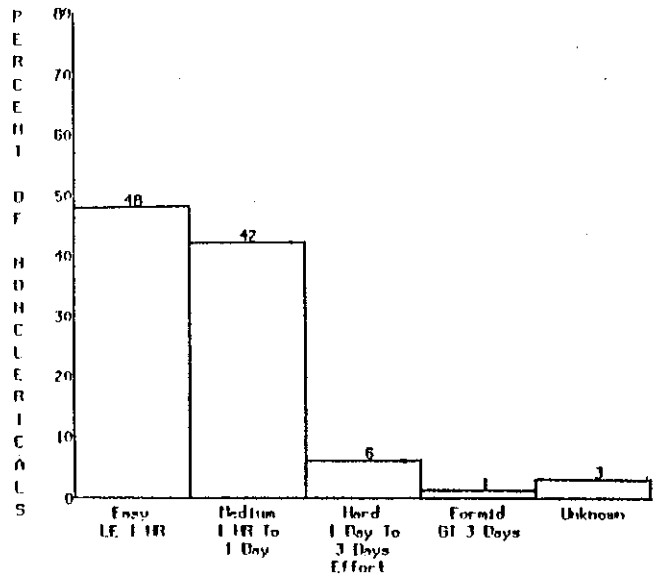
FIGURE 9 EFFORT TO CHANGE MODIFICATIONS



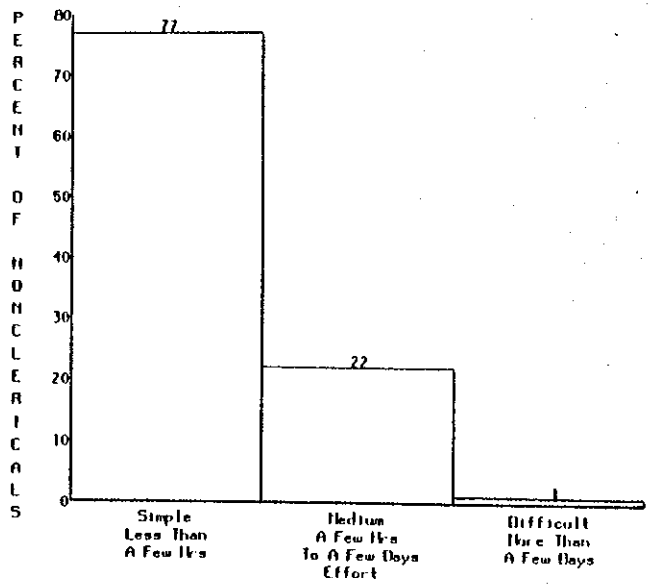
SEL1 Effort to Design Change



SEL2 Effort to Design Change

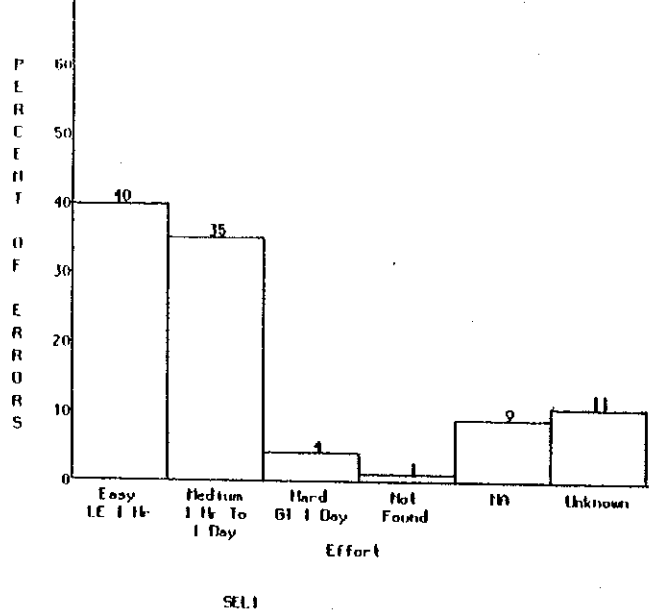


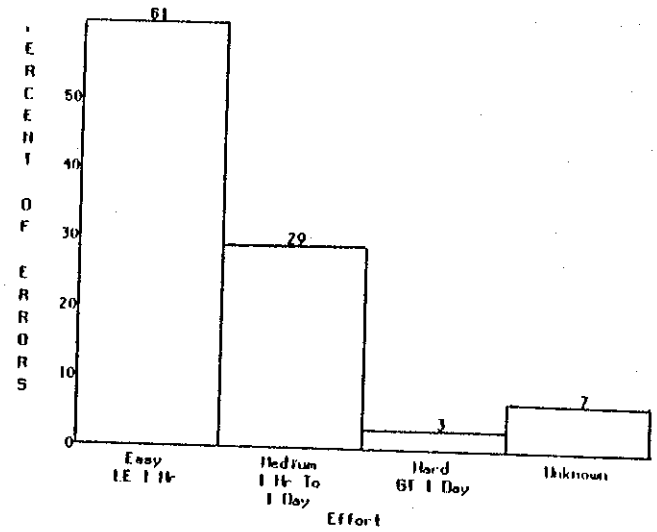
SEL3 Effort to Make Change



AIF Effort to Fix

FIGURE 10 EFFORT TO CHANGE
NONCLERICAL ERRORS





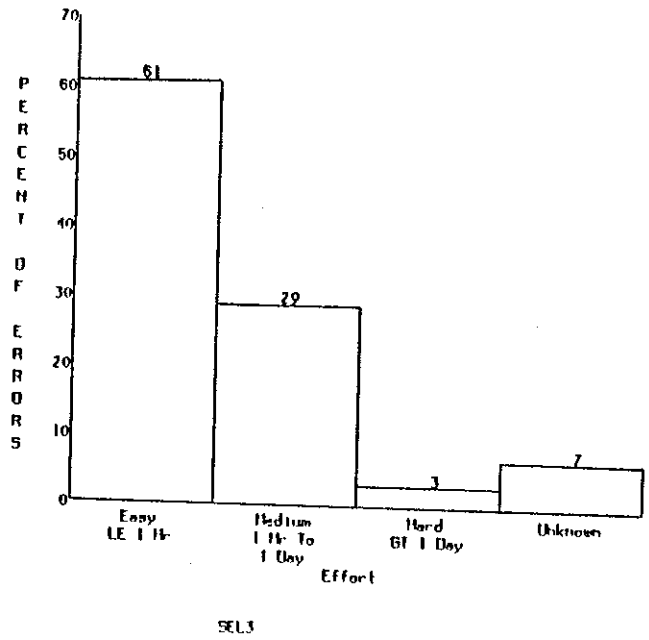
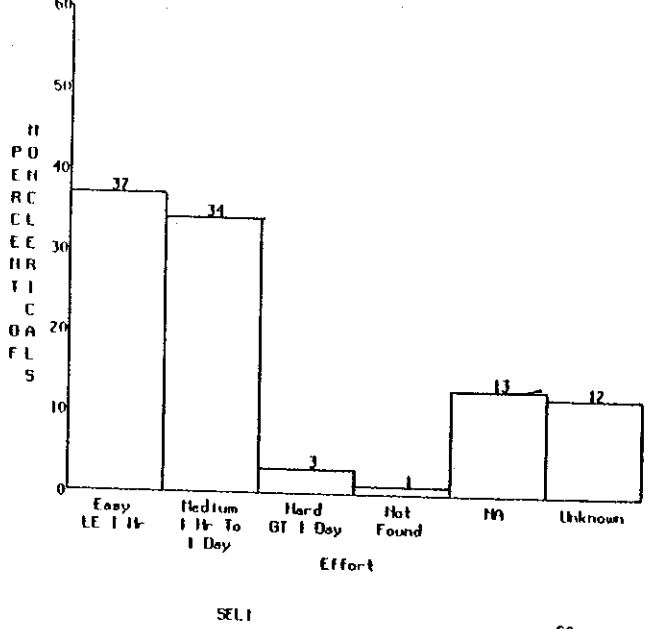
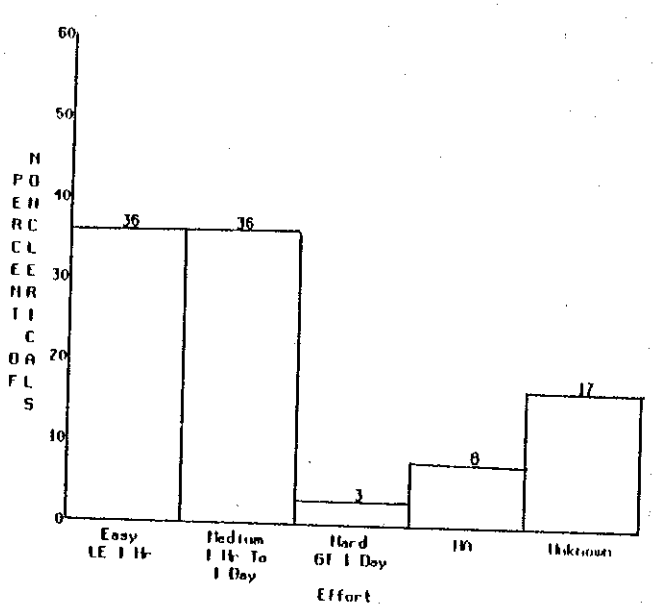


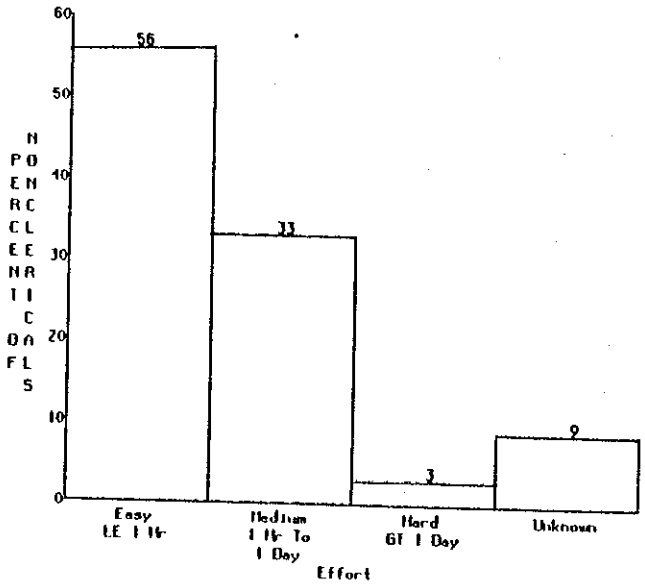
FIGURE 11 EFFORT TO ISOLATE ERROR CAUSE
ALL ERRORS



SEL1

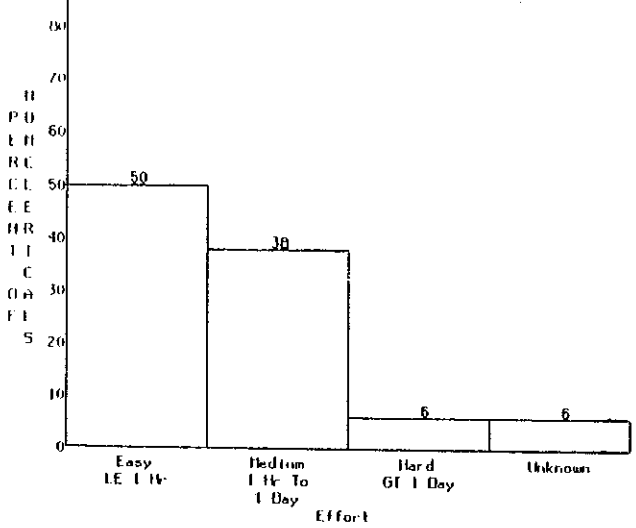


SEL2

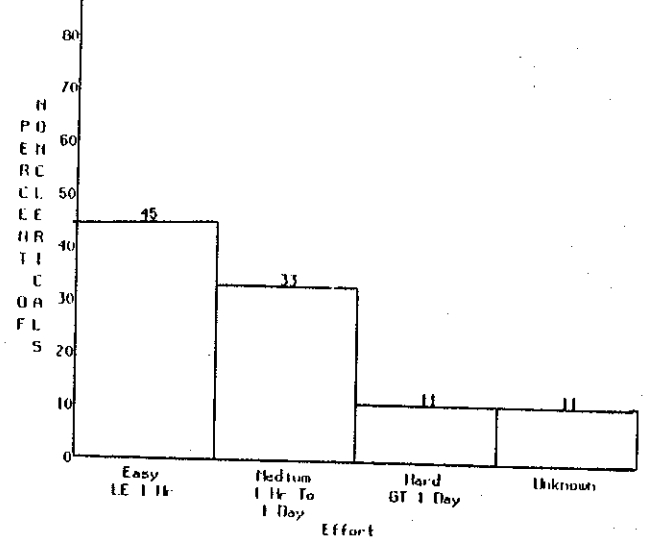


SEL3

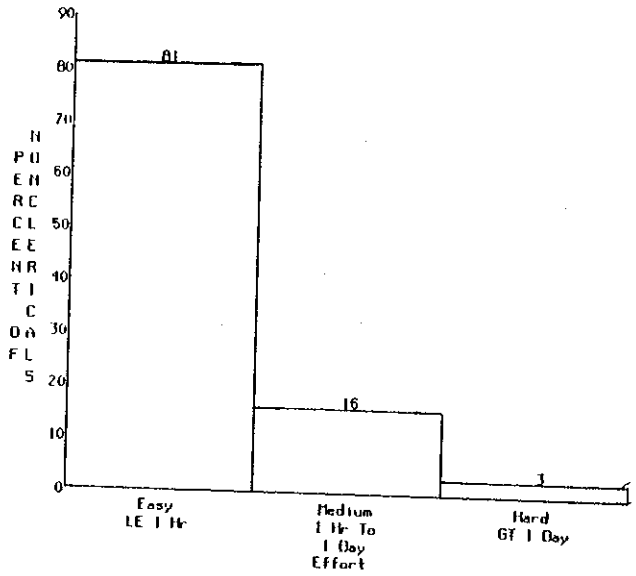
FIGURE 12 EFFORT TO ISOLATE ERROR CAUSE
NONCLERICAL ERRORS



SEL1

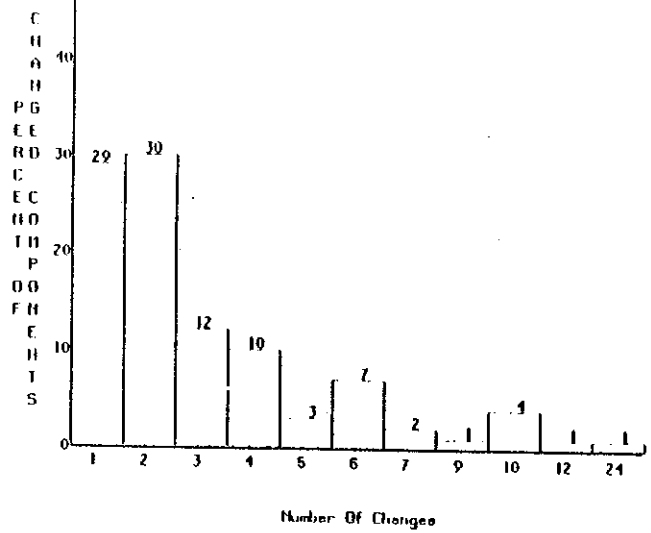


SEL2

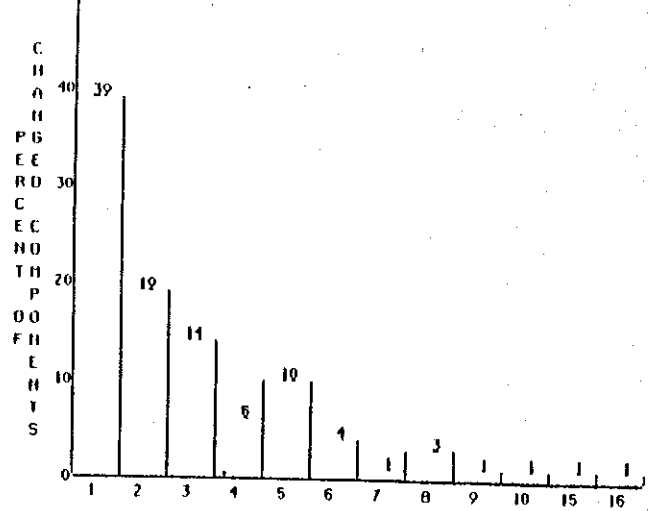


SEL3

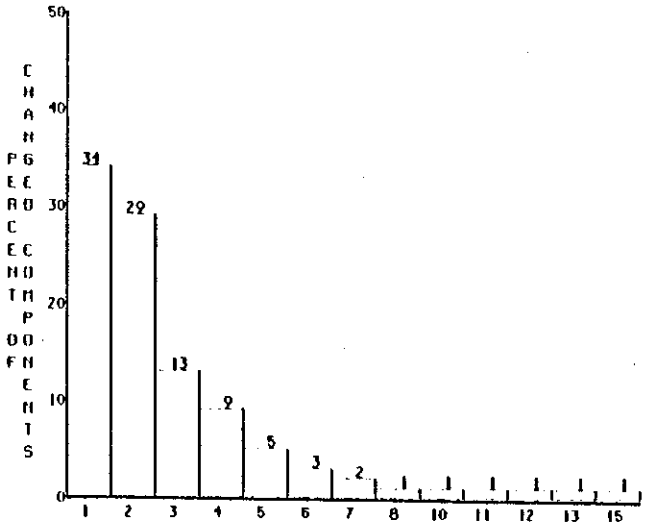
FIGURE 13 EFFORT TO ISOLATE ERRORS CAUSE CLERICAL ERRORS



SEL1

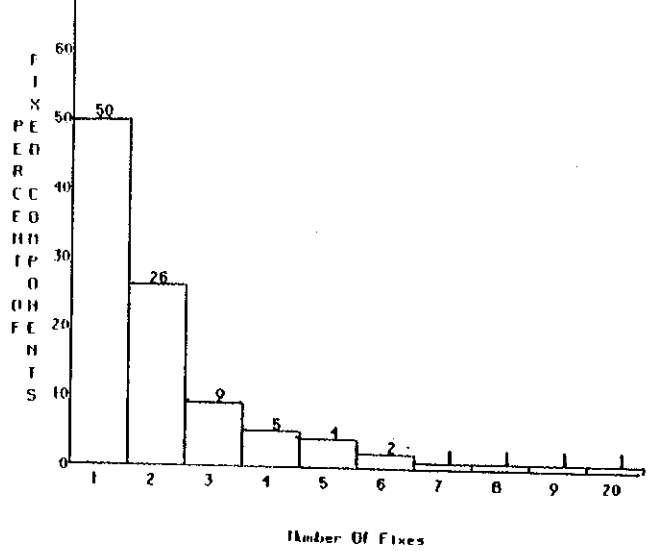


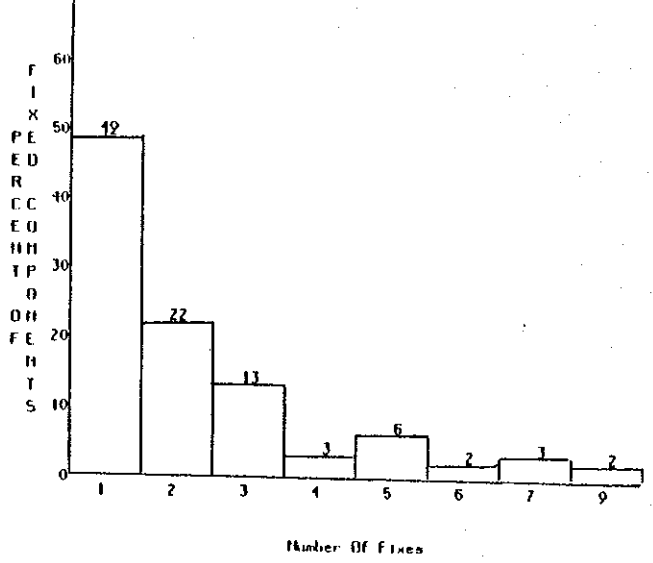
SEL2

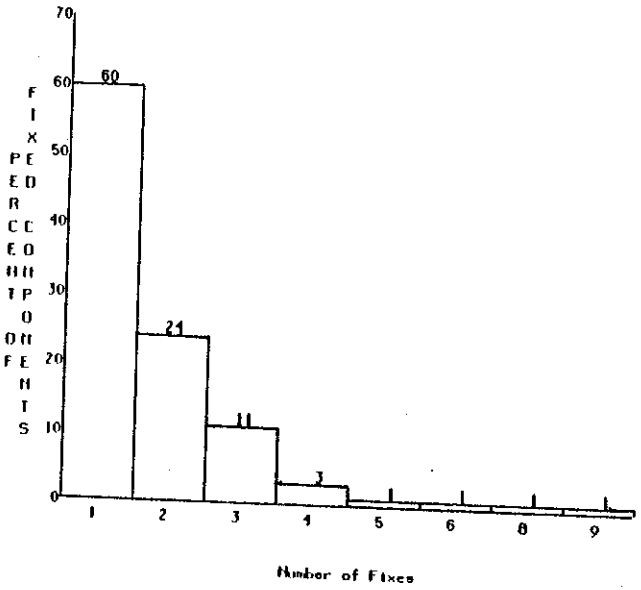


SEL3

FIGURE 14 FREQUENCY DISTRIBUTION OF CHANGES







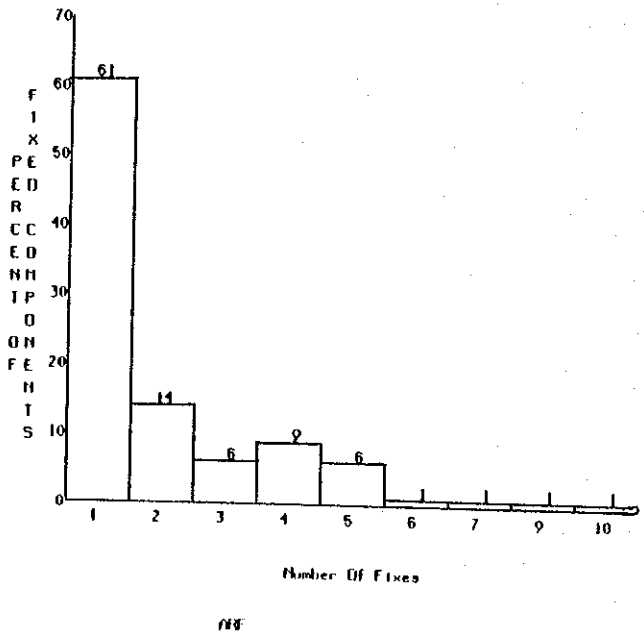
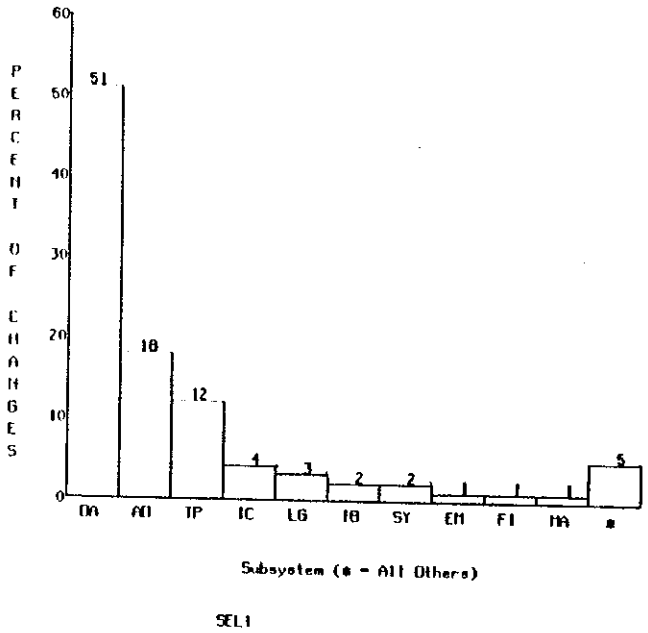


FIGURE 15 FREQUENCY DISTRIBUTION OF FIXES



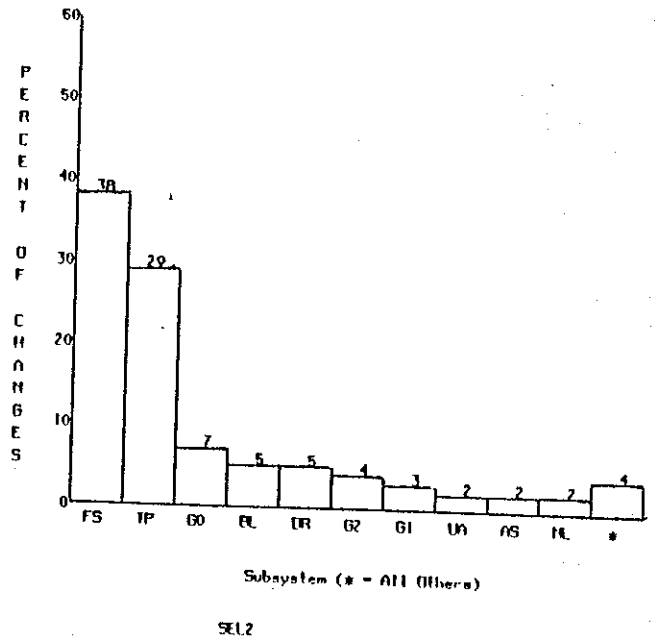
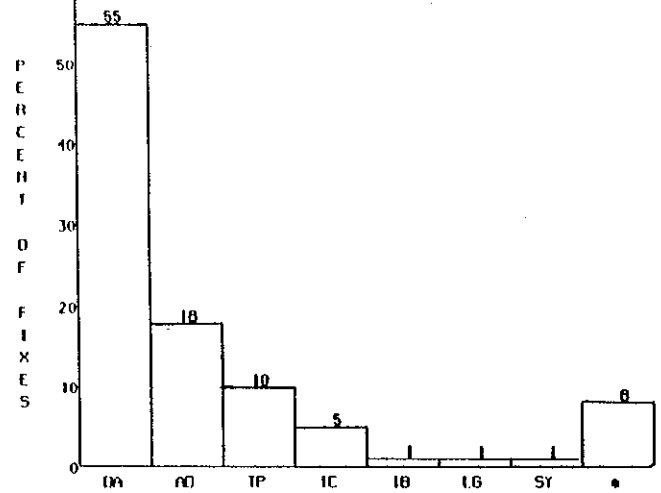
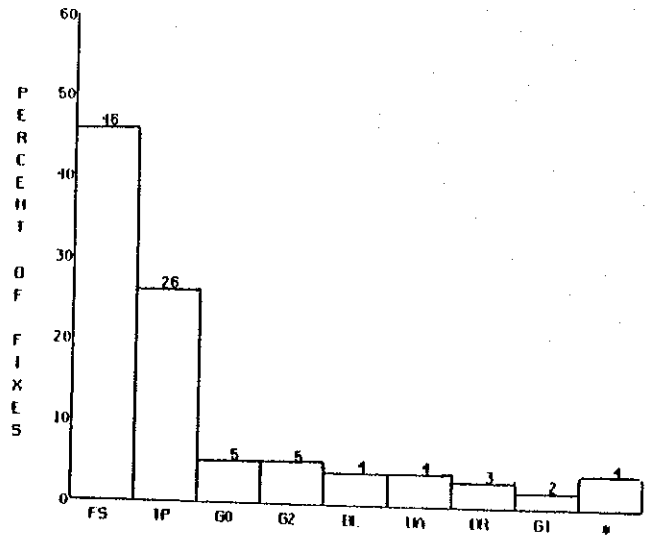


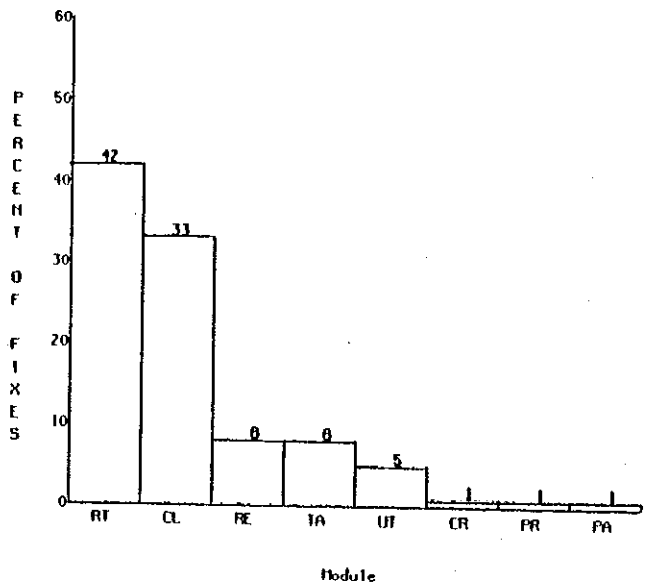
FIGURE 16 CHANGES BY SUBSYSTEM



SEL1 Fixes By Subsystem

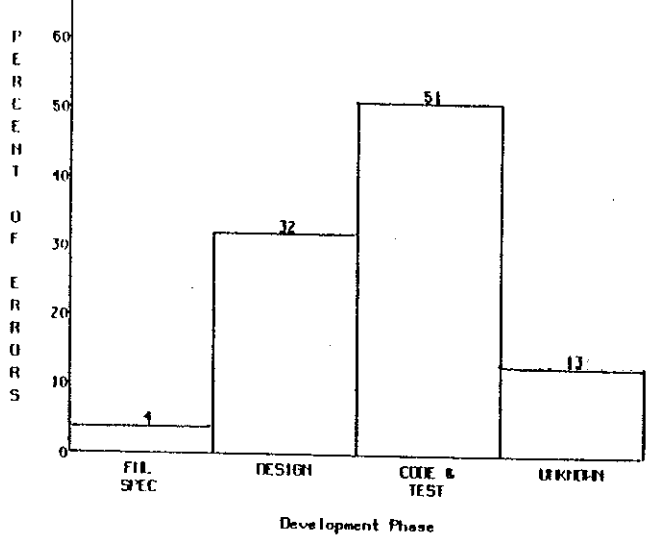


SEL2 Fixes By Subsystem

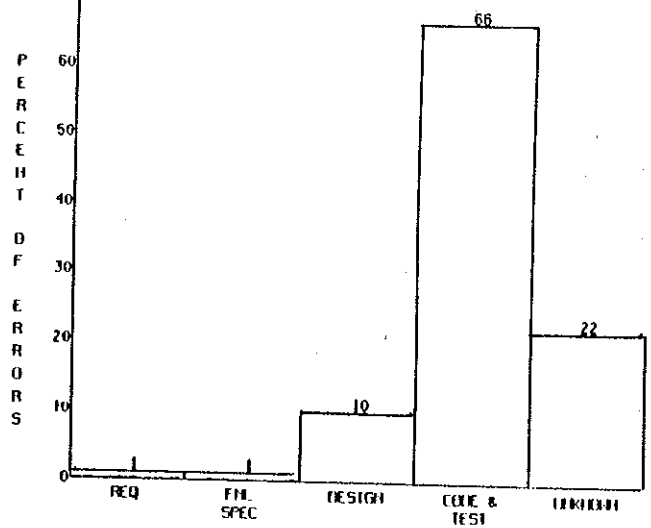


ARF Fixes By Module

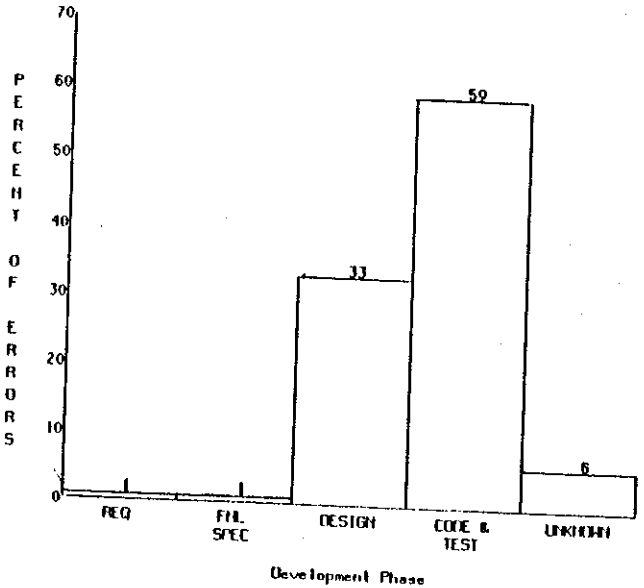
FIGURE 17



SEL1

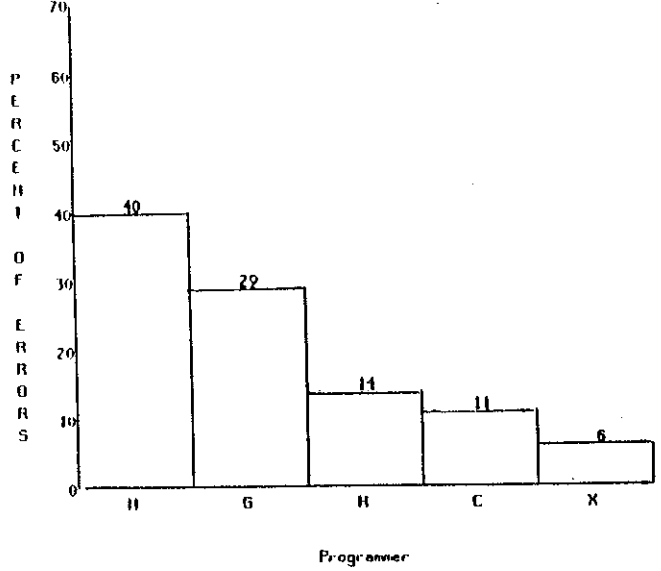


SEL2

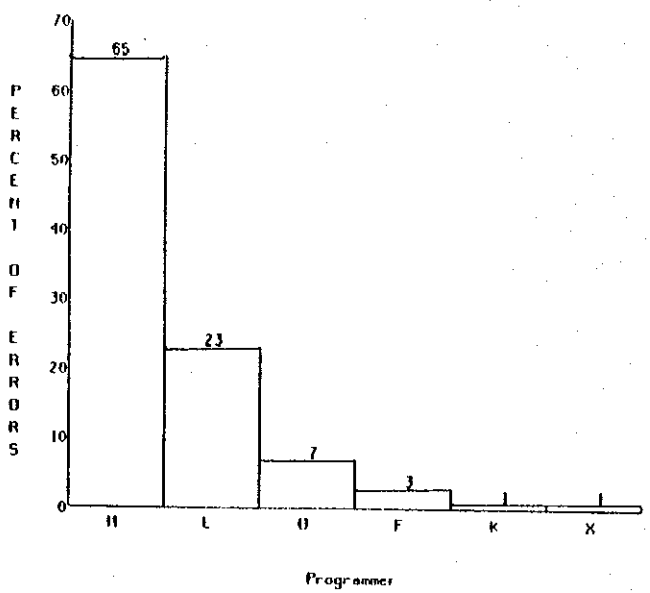


SEL3

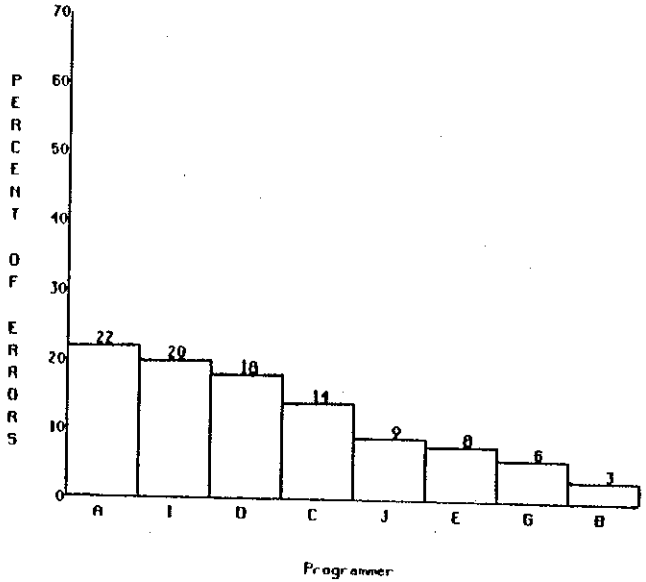
FIGURE 18 PHASE OF ERROR ENTRY



SEL1

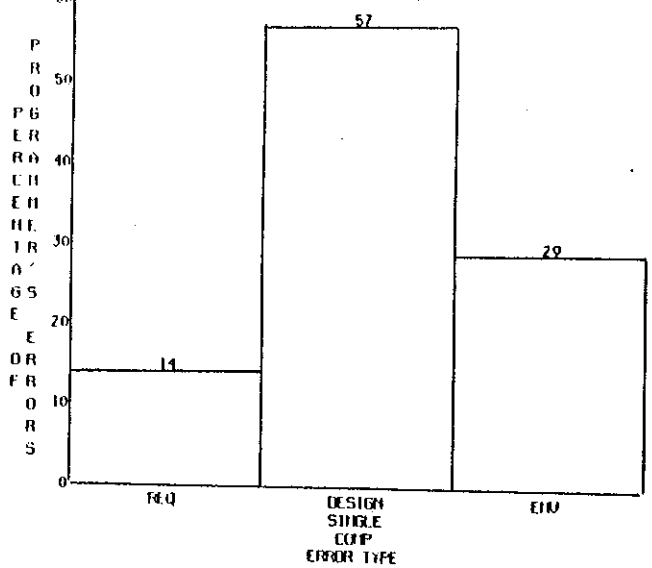


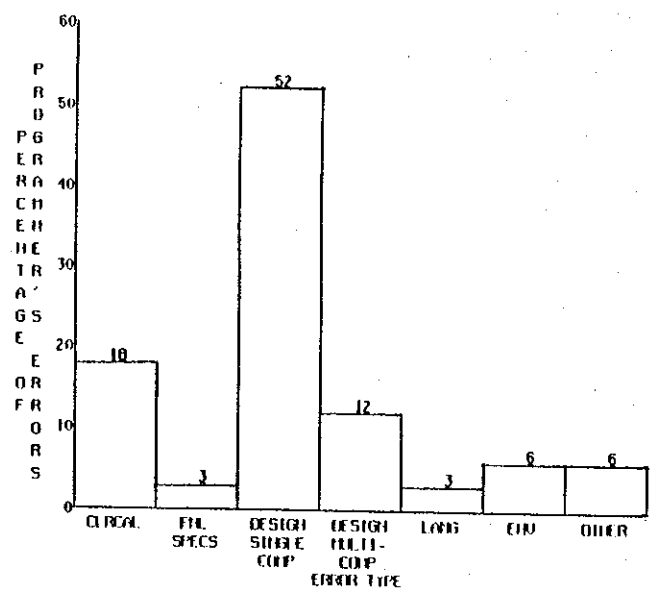
SEL2

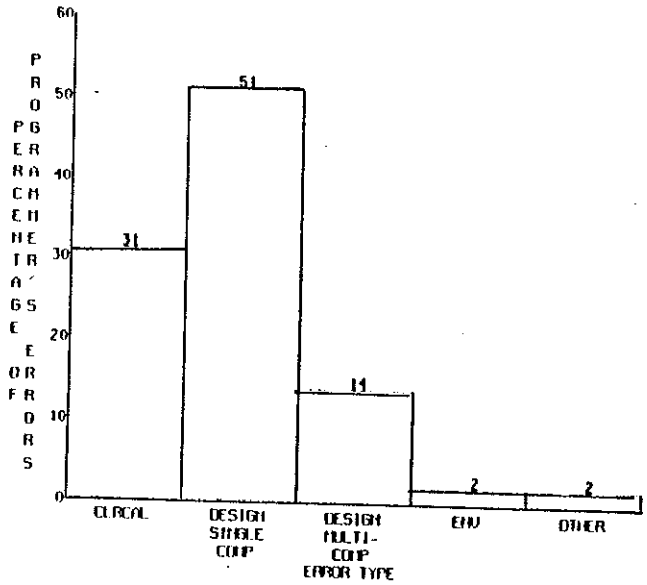


SEL3

FIGURE 19 ERRORS BY PROGRAMMER


PROGRAMMER B ERRORS
PROGRAMMER HOURS
ERROR TYPE
REQ
DESIGN SINGLE COMP
ENV
14
57
29


PROGRAMMER C ERRORS
PROGRAMMER HOURS
ERROR TYPE
CLRCAL
FIL SPECS
DESIGN SINGLE COMP
DESIGN MULTI-COMP
LANG
ENV
OTHER
18
3
52
12
3
6
6


PROGRAMMER D ERRORS
PROGRAMMER HOURS
ERROR TYPE
CLRCAL
DESIGN SINGLE COMP
DESIGN MULTI-COMP
ENV
OTHER
31
51
14
2
2

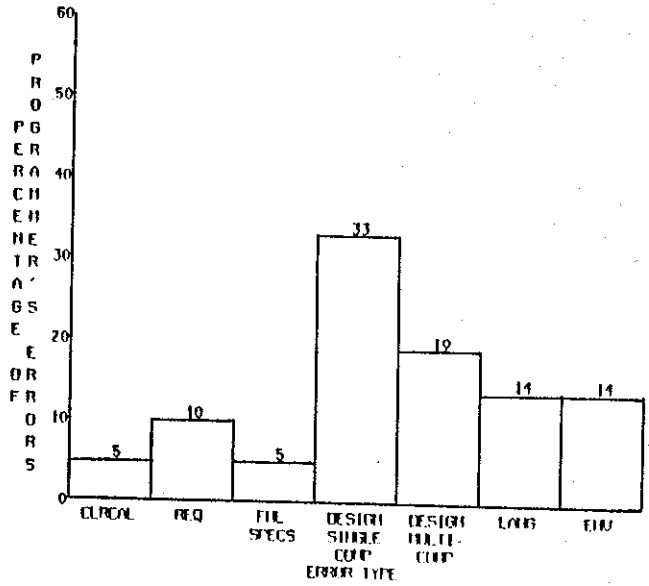
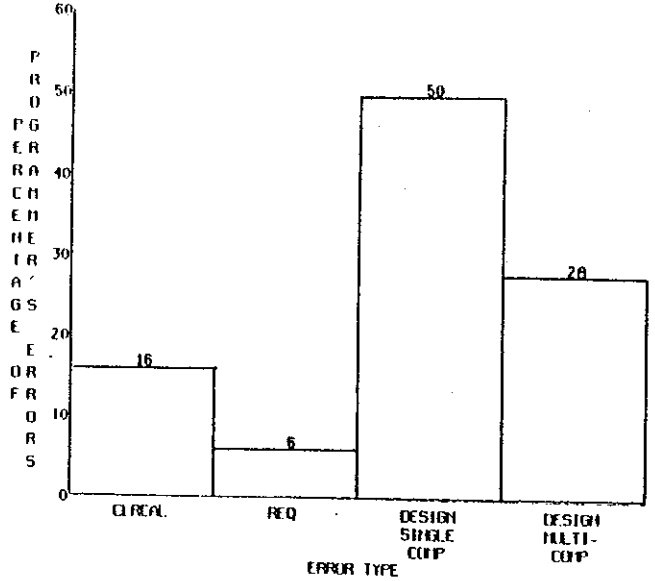
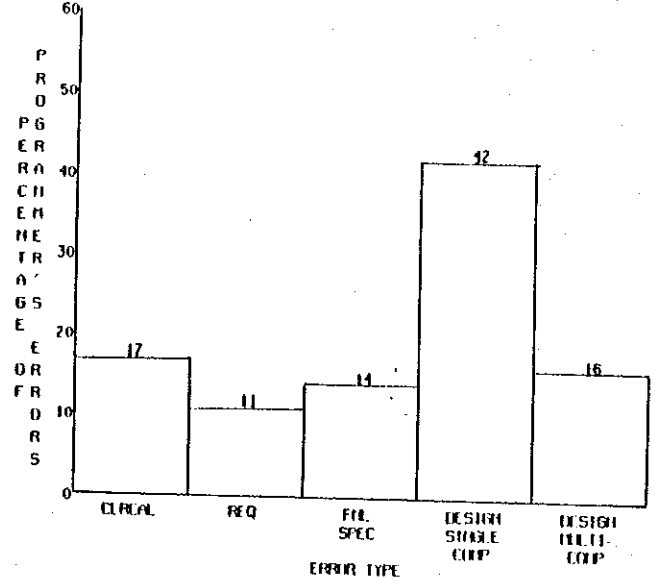

PROGRAMMER E ERRORS
PROGRAMMER HOURS
ERROR TYPE
CLRCAL
REQ
FIL SPECS
DESIGN SINGLE COMP
DESIGN MULTI-COMP
LANG
ENV
5
10
5
33
19
14
14

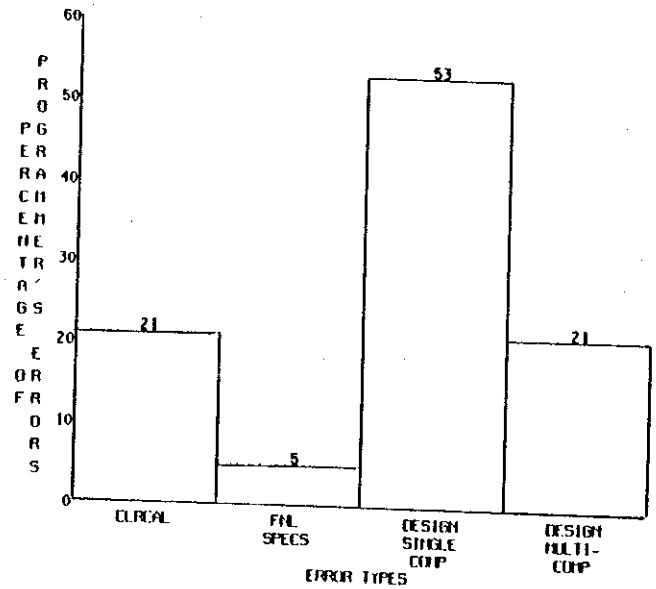
FIGURE 20



PROGRAMMER G ERRORS



PROGRAMMER I ERRORS



PROGRAMMER J ERRORS

FIGURE 20 (CONTINUED)