Evaluating Software Design Processes by Analyzing Change Data Over Time

L. J. CHMURA, A. F. NORCIO, AND T. J. WICINSKI

Abstract—**This paper presents analyses of *early* design and code change data from the Software Cost Reduction (SCR) project, a well-reported effort conducted at the Naval Research Laboratory from 1978 to 1988. The analyses are mostly time-based studies of the change data and relationships between the data and SCR personnel activity data. This analytical approach seems to allow useful insights into software design processes even when data are limited to a single software project. It also enables project personnel to notice favorable or unfavorable patterns with respect to project goals during the course of the project.**

Some analyses of the change data show patterns consistent with a major goal of the SCR project—the design and development of easy-to-change software. Specifically, most changes took a day or less to uncover and resolve; the majority of changes updated at most one module. Moreover, these percentages remained fairly stable. Also, no positive relationship appeared between error-correction effort and the number of days that an error remained in the SCR design documentation. Other analyses suggest that consistency may have been temporary. For example, the analyses suggest a stepwise growth in average change effort, and an increasing percentage of changes resulted in module interface updates.

Certain specific ratios between SCR change data and personnel activity data show promise as possible indicators of design incompleteness. The ratios are based on data of the kinds that are typically collected on software projects.

Index Terms—**Data collection, Software Cost Reduction project, software design.**

I. Introduction

BASILI and Weiss describe a methodology for collecting valid software engineering data [2]. The intent is to capture data that can yield insights into software development and maintenance processes, that help confirm or reject claims made for different software engineering technologies, and that point to better techniques for prevention, detection, and correction of errors. Since the 1970's, their methodology has been applied to a few projects at the Naval Research Laboratory (NRL). The application has been limited for a number of reasons. One is that such data collection tends to be time consuming and costly. Indeed, a major effort can add as much as 5–15% overhead to a project [9]. A second reason is that there is a major limitation to the goal directed data collection approach in a actual development environments—the inability to isolate the effects of single factors. As a consequence, project managers have been less than enthusiastic about data collection.

A result of the limited application is that we have data for a few projects that differ greatly in staffings, goals, and applications. Further, for some projects, our data are incomplete because the collection efforts were terminated before project completion. This has caused difficulties in analyzing and reporting on the data; for example, we often cannot generate summary statistics at the end of a project and compare them with similar statistics from past projects. Furthermore, when we do produce summary statistics, they have provided us with little insight into the software design processes and have proven difficult to compare with similar statistics published in the open literature.

An approach we have adopted for dealing with these difficulties is to view and analyze software engineering data over time. Often, time-based measures allow project personnel to detect favorable and unfavorable trends even before project completion. An added advantage is that the underlying data sets can be subjected to statistical techniques that can highlight trends and potential relationships between measures. As part of our approach, we have established some guidelines for presentation of such analyses. One guideline is to avoid a jittery graph by plotting the cumulative value of a measure instead of its incremental values, which tend to vary greatly between time periods. A second is to avoid changing the historical pattern of a graph associated with change in a measure's range by plotting a measure as a percentage of the total. A percentage plots is, of course, a special kind of ratio plot. Accordingly, a third guideline is to encourage comparison between different components of a software design and to highlight relationships between different data by examining ratios between data, specifically change data and personnel activity data. In short, our experience has reemphasized the importance of observing and understanding system *dynamics* as an approach to understanding software development and evolutionary processes that Belady and Lehman [3] discussed in the 1970's and that has been illustrated more recently by Grady in 1987 [16].

This paper illustrates the above ideas and the time-based approach to the analysis of software engineering design change data. It presents analyses of design changes proposed and made by software development engineers who worked on the Software Cost Reduction (SCR) project at

Manuscript received January 13, 1988; revised February 15, 1990. Recommended by N. Schneidewind.

L. J. Chmura and T. J. Wicinski are with the Naval Research Laboratory, Human-Computer Interaction Laboratory, Washington, DC 20375.

A. F. Norcio is with the Department of Information Systems Management, University of Maryland, Baltimore County, Catonsville, MD 21228, and the Naval Research Laboratory, Human-Computer Interaction Laboratory, Washington, DC 20375.

IEEE Log Number 9035724.

NRL. There are five sections in the paper. The remainder of this section contains a brief overview of NRL's SCR project and Software Technology Evaluation project. The second section is a description of the techniques and strategies that were used in collecting and categorizing the data. The third section is a detailed discussion of the change and error data. The final two sections contain the analyses of the data and their possible implications.

A. The Software Cost Reduction Project

The Software Cost Reduction Project began in 1978 at NRL as a cooperative effort with the Naval Weapons Center (NWC). The purpose was to redevelop version 2 of the Operational Flight Program for the A-7E aircraft using improved software technology [10]. Two major goals were to 1) demonstrate the feasibility of using selected software engineering techniques in developing complex, real-time software, and 2) provide a model for later NWC software designers [24]. Software engineering techniques such as formal requirements specification [19], information hiding [22], abstract interfaces [23], and cooperating sequential processes [13] were prominent among the technologies applied. The claimed advantage of these technologies was that they facilitate the development of software that is easy to change and maintain.

A complete discussion of the project's software requirements was provided by Heninger *et al.* [17]. Britton and Parnas provided a detailed description of the module design structure [5]. Fig. 1 presents an example of a module interface specification (i.e., a *design* specification) taken from a specification for the device interface module [21]. A standard organization for such specifications was described by Clements *et al.* [11].

The SCR project terminated at the end of 1987 after implementing three subsets of the operational flight program requirements. The subsets were evaluated and tested using ground using ground-based test facilities at NWC.

B. The Software Technology Evaluation Project

The data reported here was collected and analyzed by researchers working on the Software Technology Evaluation (STE) project, which was an NRL project separate from the SCR project in terms of goals, staffing, and funding.[1] The goal of the STE project was to evaluate alternative software development technologies. A major task of the STE project, therefore, was to provide the basis for an objective evaluation of the methodology used in the SCR project.

The approach followed in the STE project was to monitor, evaluate, and compare software development technologies used in different software projects. The monitoring and evaluating processes consisted of goal-directed data collection and analyses techniques [2]. For the SCR project, data was collected in three areas: personnel activity [20], changes to requirements [8], and changes to design and code.

The STE project terminated in the mid-1980's.

DLVI: VISUAL INDICATORS (Auto-Cal and Non-Align Indicators)

1. Introduction

There are two visual indicators controlled by the OFP on the A-7E aircraft; one that can, and one that cannot, be seen by the pilot during flight. These are currently labeled "IMS Non-Aligned" and "Auto-CAL", respectively. Each can be on steady, on blinking, or off.

2. Interface overview

2.1 ACCESS PROGRAM TABLE

Program	*Parameters*	*Description*	*Undesired events*
+G/S_AUTOCAL_INDICATOR+	p1: VIS_ind_cntrl; O/I	!+Auto-cal+!	None
+S_AUTOCAL_BLINK_RATE+	p1: real; I	blink/sec	
+G/S_NON_ALIGN_INDICATOR+	p1: VIS_ind_cntrl; O/I	!+Non-align+!	
+S_NON_ALIGN_BLINK_RATE+	p1: timeint; I	!!rate!!	

Effects

+S_AUTOCAL_INDICATOR+	IF p1=On THEN "Auto-Cal" indicator turned on; IF p1=Off THEN "Auto-Cal" indicator turned off; IF p1=$Intermittent$ THEN "Auto-Cal" indicator turned on and off at the rate set by +S_AUTOCAL_BLINK_RATE+, or at the system default rate.
+S_AUTOCAL_BLINK_RATE+	When commanded to blink, the blink !!rate!! will be p1.
+S_NON_ALIGN_INDICATOR+	IF p1=On THEN "Non-Align" indicator turned on; IF p1=Off THEN "Non-Align" indicator turned off; IF pa=$Intermittent$ THEN "Non-Align" indicator turned on and off at the rate set by +S_NON_ALIGN_BLINK_RATE+, or at the system default rate.
+S_NON_ALIGN_BLINK_RATE+	When commanded to blink, the blink !!rate!! will be p1.

3. Local type definitions

VIS_ind_cntrl	Enumerated On, Off, $Intermittent$

4. Dictionary

!+Auto-cal+!	The state of the auto-cal indicator as last set by +S_AUTOCAL_INDICATOR+
!+Non-align+!	The state of the non-align indicator as last set by +S_NON_ALIGN_INDICATOR+

5. Undesired event dictionary None

6. System generation parameters

#Autocal blink default#*	Type: timeint. Default blink interval for "Auto-Cal" indicator.
#Auto-Cal init state#*	Type: VIS_ind_cntrl. The system-load-time value for !+Auto-cal+!
#Nonalign blink default#*	Type: timeint. Default blink interval for "Non-Align" indicator.
#Non-Align init state#*	Type: VIS_ind_cntrl. The system-load-time value for !+Non-align+!

* The value of the system generation parameter may be set by user software. See section 2.2 of the introduction to this document.

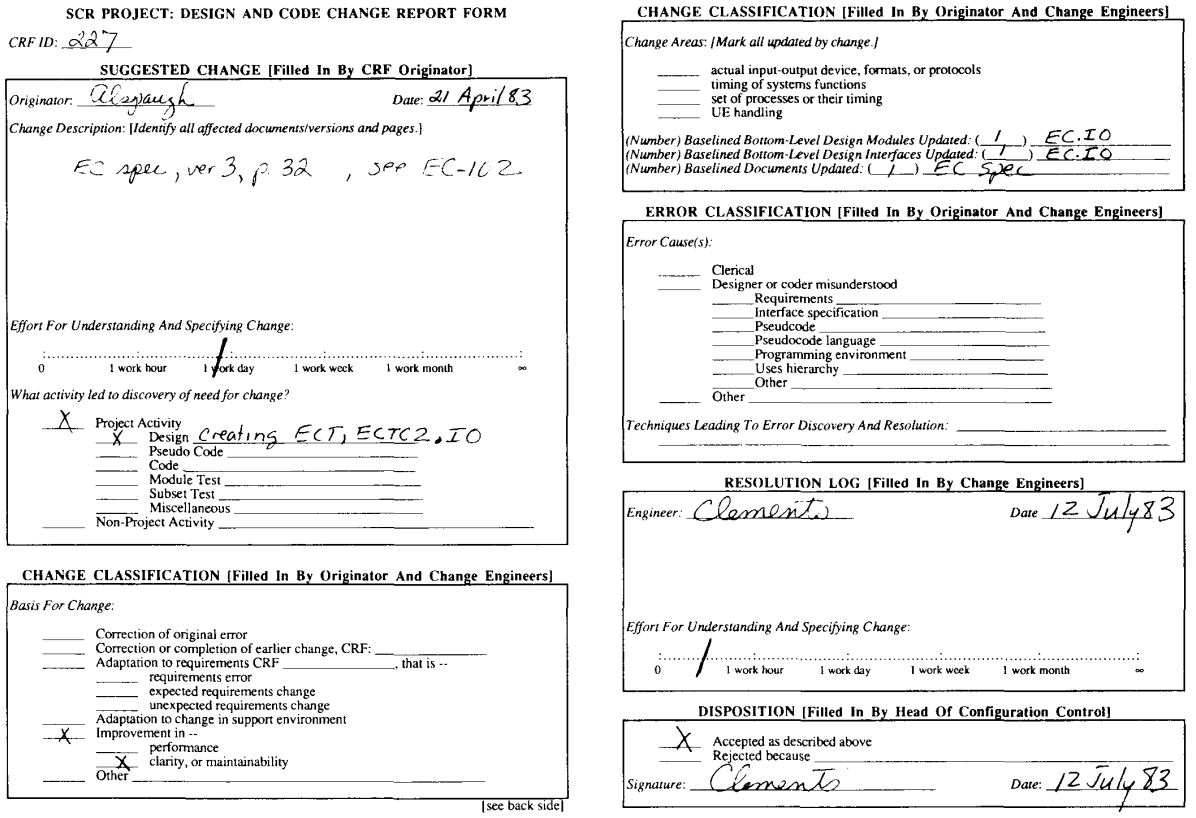
Fig. 1. Example module interface specification.

II. Collection of Change Data

From 1980 until early 1985, SCR project engineers reported design and code problems, suggested design changes, and logged their modification activity to *baselined* (i.e., published and change-controlled) interface specifications, pseudocode, and TC2 code[2] on Change Report Forms (CRF's). An example of a completed CRF is presented in Fig. 2. There were two reasons for this procedure. First, it was required as part of the SCR project's configuration management (CM) procedures. Second, such data were needed by STE researchers for eval-

[1]The project was at one time funded by the DoD STARS Program as Measurement Area Task G-06.

[2]TC-2 code is the assembly language code for the IBM System 4 PI model TC-2 computer. The A-7E Operational Flight Program runs on this machine.



SCR PROJECT: DESIGN AND CODE CHANGE REPORT FORM

CRF ID: 227

SUGGESTED CHANGE [Filled In By CRF Originator]

Originator: Heyaugh Date: 21 April 83

Change Description: [Identify all affected documents/versions and pages.]

EC spec, ver 3, p. 32, see EC-102

Effort For Understanding And Specifying Change:

0 | 1 work hour | 1 work day | 1 work week | 1 work month | ∞

What activity led to discovery of need for change?

X Project Activity
X Design creating EC1, EC1C2, IO
___ Pseudo Code
___ Code
___ Module Test
___ Subset Test
___ Miscellaneous
___ Non-Project Activity

CHANGE CLASSIFICATION [Filled In By Originator And Change Engineers]

Basis For Change:

___ Correction of original error
___ Correction or completion of earlier change, CRF: ___
___ Adaptation to requirements CRF ___, that is --
___ requirements error
___ expected requirements change
___ unexpected requirements change
___ Adaptation to change in support environment
X Improvement in --
___ performance
X clarity, or maintainability
___ Other ___

[see back side]

CHANGE CLASSIFICATION [Filled In By Originator And Change Engineers]

Change Areas: [Mark all updated by change.]

___ actual input-output device, formats, or protocols
___ timing of systems functions
___ set of processes or their timing
___ UE handling

(Number) Baselined Bottom-Level Design Modules Updated: (1) EC.IO
(Number) Baselined Bottom-Level Design Interfaces Updated: (1) EC.IO
(Number) Baselined Documents Updated: (1) EC Spec

ERROR CLASSIFICATION [Filled In By Originator And Change Engineers]

Error Cause(s):

___ Clerical
___ Designer or coder misunderstood
___ Requirements
___ Interface specification
___ Pseudocode
___ Pseudocode language
___ Programming environment
___ Uses hierarchy
___ Other
___ Other

Techniques Leading To Error Discovery And Resolution: ___

RESOLUTION LOG [Filled In By Change Engineers]

Engineer: Clements Date 12 July 83

Effort For Understanding And Specifying Change:

0 | 1 work hour | 1 work day | 1 work week | 1 work month | ∞

DISPOSITION [Filled In By Head Of Configuration Control]

X Accepted as described above
___ Rejected because ___

Signature: Clements Date: 12 July 83

Fig. 2. Completed CRF form.

uating achievement of SCR project goals. The specific design of the CRF form was based on a goal-directed data collection approach [6]. In 1985, the use of paper CRF's was replaced by a computer-based CM tool.

STE researchers validated primarily those CRF's that were *resolved* either by official acceptance and incorporation into the baselined documentation, or by official rejection of the proposed change. Ideally, validation should have been a continuing activity that occurred as CRF's were generated and resolved. Validation of SCR CRF's, however, tended to be an aperiodic activity in which large groups of CRF's were validated at one time. The validation consisted of checking completeness, accuracy, etc. It often included discussions with persons who submitted the CRF's, authors of affected documents, and SCR CM personnel. A major validation point concerned what constituted a *design or code change*. Basically, the view taken was that a change was conceptual; that is, one should have been able to state a proposed change in a simple declarative sentence and the change *may* comprise alterations to one or more baselined interface specification or implementation documents. In addition, a change that was described in one CRF similar to a change in a CRF resolved and implemented in earlier baselines (i.e., a change that required completion or correction to earlier baselined alterations) was considered a unique or *new* change. Thus, a change was to have a unique basis—error correction, adaptation to outside change, improvement, or other (see Fig. 2). The notion of basis followed the scheme presented by Swanson [27]. A proposed change that was rejected obviously resulted in no alterations.

This definition of a design or code change caused problems. Occasionally a CRF was submitted that incorporated more than one change, and different engineers sometimes submitted the same change on different CRF's. For example, it was not unusual for a CRF to describe two conceptual changes as in the following:

> "The last sentence of the description is ambiguous. Replace it with · · · Note also that the word *descriptor* is misspelled."

A workable and reasonable solution used by STE researchers for dealing with these situations was to split submitted CRF's that incorporated more than one change into an appropriate number of CRF's, such that each described a single change. Multiple CRF's that describe identical changes were consolidated into one CRF. One result of this policy was that there was not a one-to-one correspondence between *submitted* CRF's and *validated* CRF's. The other result was, of course, that there was a one-to-one correspondence between proposed changes and validated CRF's.

There were other sections of the CRF that caused difficulties. One was the basis of an accepted change. A problem was that it was not sufficient to define an error as a discrepancy between a specification and its implementation. For example, an inadequate interface design was considered an error; an adequate interface design needing enhancements was considered an improvement. The only reasonable solution to this problem was to let SCR lead engineers decide in such situations. Another problem was determining whether or not a change was a correction or completion of an earlier change that was already incorporated in a baseline. The fact was, that after a long period of time or after many versions of a document, authors frequently forget earlier changes that had addressed the same issues presented in current CRF's. For each of the CRF's reported in this study, STE researchers reviewed all versions of all documents baselined prior to resolution of the CRF and discussed all questions with lead SCR engineers. This was a laborious process but was necessary to ensure that corrections or completion errors were properly identified.

Lastly, the SCR project's CM procedures were not perfect. Validators found a few CRF's that were not resolved, but, nevertheless, were implemented in published specifications. The only reasonable solution for this was to resolve such CRF's with the date of the latest baselined specification and to submit CRF's for remaining aspects of the change. Validators also found modifications for which there were no corresponding CRF's. The policy for this was to submit CRF's and record them as immediately resolved with the date of issue of the appropriate baselined specifications.

III. Overview of Early SCR Change Data

A. General

This paper is a summary of 325 validated CRF's that were resolved by January 1984 (i.e., through the CRF's were no longer validated by STE researchers.

By January 1984, engineers had submitted 424 CRF's. The 325 CRF's reported here map to 296 (70%) of those submitted and resolved by SCR CM personnel by that date. Figs. 3 and 4 are profiles of resolution activity for the CRFs.[3] By January 1984, approximately 47 500 person hours had been expended on the SCR project. The 400 hours of resolution effort accounted for approximately 1% of project activity. Table I presents the distribution of the CRF's categorized by the originators' activities when the CRFs were generated.

A large proportion of CRF's originated during design activity. In addition, by January 1984 only 15% of SCR project hours were spent on pseudo coding, coding, and testing activities. This means the changes reviewed in this study can be characterized as changes that are typically proposed and made early in software development, in contrast with changes reported elsewhere [1], [15], [29].

[3]These figures, together with many of the following figures, are plots of cumulative data.

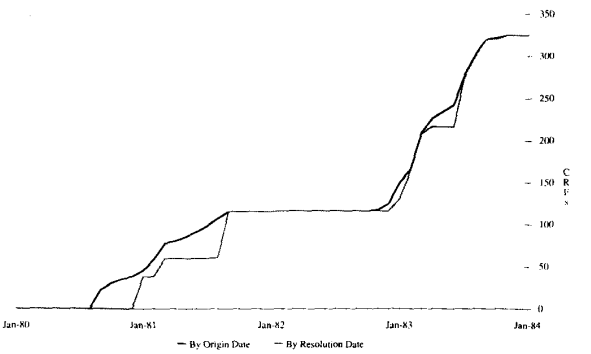


Fig. 3. CRF accumulation.

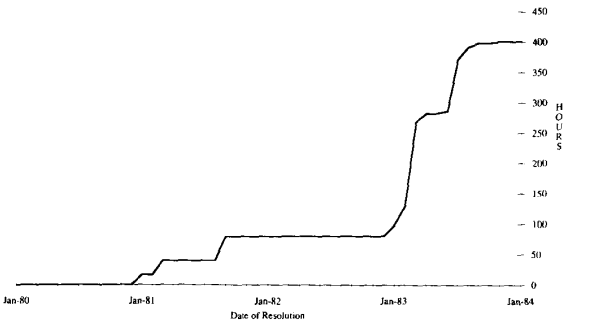


Fig. 4. Cumulative effort in resolving CRF's.

TABLE I
Activities Leading to CRF Origination

Project Activity:	
Design (e.g., Module Interface Specification)	209 (64%)
Pseudo Code	53 (16%)
Code	1 (0%)
Test	26 (8%)
Misc	15 (5%)
Unknown	5 (2%)
Total:	309 (95%)
Non-Project Activity (e.g., CRF validation):	
Total:	16 (5%)

Twenty-eight (9%) of the 325 proposed changes were rejected; this required approximately 18 hours (4%) of the total hours expended on the changes (see Figs. 5 and 6). The 9% figure is small compared to the 37% figure reported by Day for major maintenance updates to an operational Army command and control system [12]. It is also smaller than the 20% figure reported by Shooman and Bolsky for errors discovered and corrected during test and integration of a modest-size control program at Bell Telephone Laboratories [26]. The 4% effort figure is comparable to the 3% figure reported by Day. Care must be taken with these comparisons, however. These figures are from different times in different project life cycles, and it is not clear that there is a common definition of change. More important, SCR requirements changes were a separate SCR CM concern and were not incorporated in the data reported here [8].

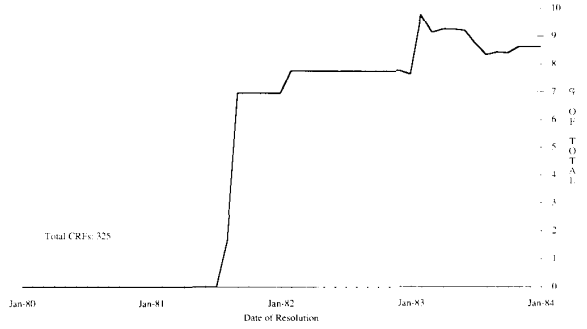


Fig. 5. Rejected CRF's: percentage of total.

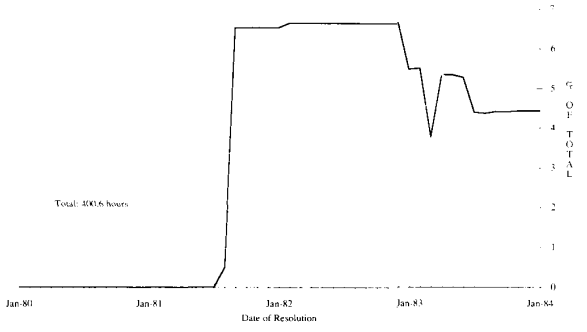


Fig. 6. Rejected CRF resolution effort: percentage of total.

The remaining 297 accepted CRF's resulted in modifications to 47 baselined module interface specifications, most of which are packaged in two documents. No module implementation documents (which include psuedocode) or code were affected for the simple reason that none were baselined prior to January 1984. This limit of impact to interface specifications means that the 297 changes can be further characterized as early *design* changes.

The bases for the 297 accepted changes are presented in Table II. None of the changes were the result of changes to the software requirements specification. This can probably be attributed to the following:

1) an extensive requirements specification was generated prior to design [17],

2) the requirements specification has been shown to be relatively error free and remarkably free of ambiguities [8],

3) as noted earlier, the changes reported can be characterized as early changes, and

4) the SCR project is redeveloping software for a fixed operational version of the A-E flight software.

The percentage of error corrections (see Table II and Fig. 7) is higher than the range (40–64%) reported Basili and Weiss [1], [29]. But it is far lower than the 96% figure reported by Shooman and Bolsky [26] and is decreasing. The proportion of total CRF effort spent on error corrections (Fig. 8), even though decreasing, sharply contrasts with the 17% figure reported by Lientz and Swanson [18] for commercial data processing software maintenance efforts, and the 21% figure reported by Day [12]. It should

TABLE II
BASES OF ACCEPTED CRF'S

Error Corrections:		
Original		144 (48%)
Continuation or Completion		55 (19%)
	Total:	199 (67%)
Modifications:		
Adaptation to requirements change		0 (0%)
Adaptation to support environment change		0 (0%)
Improvement in performance		2 (1%)
Improvement in clarity		89 (30%)
Other		7 (2%)
	Total:	98 (33%)

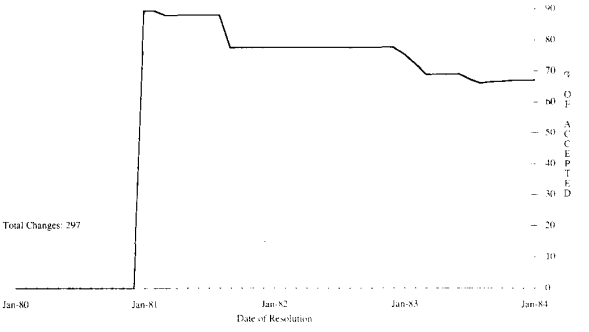


Fig. 7. Error corrections: percentage of accepted changes.

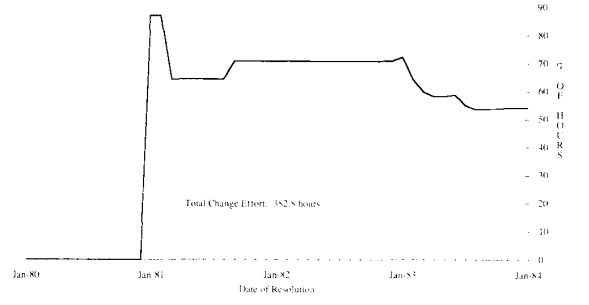


Fig. 8. Error correction effort: percentage of accepted CRF resolution effort.

be noted again, however, that the SCR requirements document change data are not included in this summary.

The proportion of error corrections that involved completing or correcting a prior change (see Fig. 9) is large as compared to the 6–12% range of figures reported by others [1], [28], [29] and seems to be increasing in a step fashion. The 12% figure is computed from data presented by Weiss [28] and by Weiss and Basili [29]. This large proportion could be the result of the many hours spent by STE and SCR engineers in assuring the correct identification of correction and completion errors.

B. The SCR Ease-of-Change Goal

A major objective of the SCR project was to produce software design, code, and a documentation set that could be used to scope and to implement changes easily. The SCR design and code change CRF was designed explic-

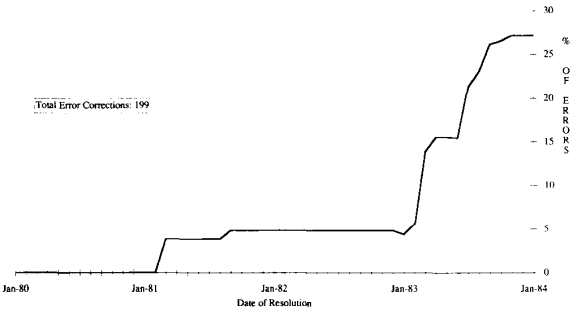


Fig. 9. Correction or completion errors: percentage of error corrections.

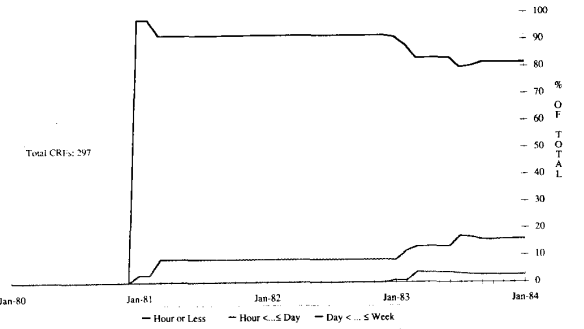


Fig. 10. Accepted CRF's categorized by resolution effort.

itly to collect data to try to evaluate achievement with respect to this objective.

Fig. 10 presents the distribution of effort required for understanding and incorporating the 297 accepted changes into the SCR project's design documentation set; Fig. 11 presents the distribution for error corrections only. Only one of the 28 rejected CRF's was not implemented because the proposed change was deemed not worth the effort. Most changes (81%) took an hour or less to understand and resolve; 98% took a day (i.e., 8 person hours) or less. Eighty-six percent of the error corrections took an hour or less to understand and resolve; 99% took a day or less. Although the data presented in Figs. 10 and 11 exhibit downward trends, these data seem to suggest that, for early changes and error corrections, SCR engineers were meeting their major objective. For errors uncovered and corrected late in the life cycle of a NASA/Goddard Software Engineering Laboratory project, Basili and Pericone [1] report 36% of the error corrections took an hour or less; 55% took a day or less. For errors uncovered and corrected late in the Wuhan University Problem Analysis Diagram Translator project, Xu reports 24% of the error corrections project took an hour or less; and 80% took a day or less [30].

Fig. 12 presents the cumulative average effort for all SCR changes and error corrections. There appeared to be a step growth in cumulative average change effort as the SCR project proceeded. This is consistent with Boehm's data that show an exponential growth in cost to fix or change software for successive phases of the software life cycle [4]. Although consistent, the average change effort for the early SCR design changes nevertheless seems quite small. Fig. 13 presents the effort for an error correction based on number of days that the error was in the system. The figure "days in system" is the difference between CRF resolution date and the earliest issue date for the interface specifications containing the error. Boehm's data imply that the longer an error remains undetected and uncorrected in a system, the greater the cost of the eventual error correction. Surprisingly, this effect does not appear in the SCR data; the correlation between days in system and average effort is 0.07, which is not significant at the 0.05 level. There may be several reasons for this. The first is that SCR requirements change data are not in-

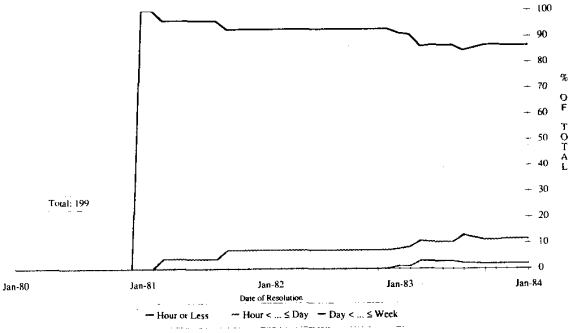


Fig. 11. Error corrections categorized by resolution effort.

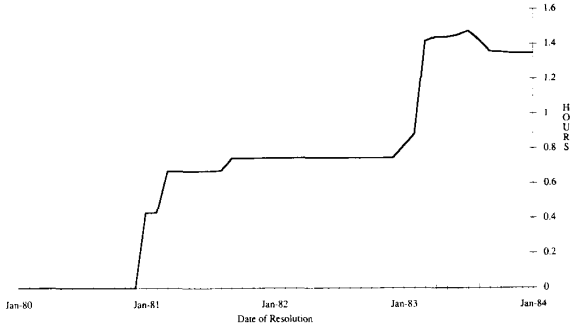


Fig. 12. Cumulative average CRF effort.

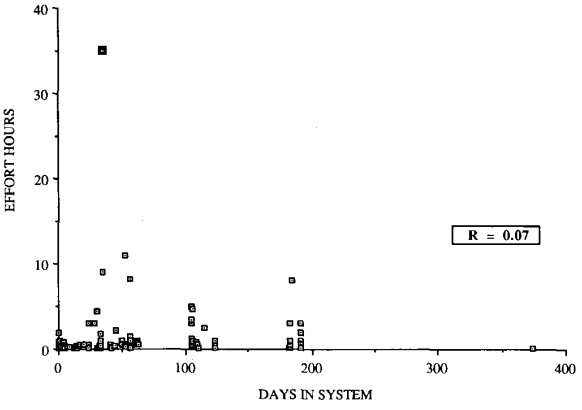


Fig. 13. Duration of an error in the system.

cluded here. The second is that the changes reported here can be considered to be only design-phase changes, and more of the SCR project's life cycle might have had to pass before any relationship appeared. The third is that there were many very low effort changes. And the fourth, of course, is that the SCR methodology may, indeed, have lessened the impact of long-term unresolved errors!

The information hiding principle was used in the SCR project for identifying and specifying a hierarchy of design modules [25]. A module was supposed to *hide* a likely changeable aspect of the A-7E flight software. This meant that a module's interface specification must be written such that the hidden information was not revealed; that is, a module's hidden information was available only to the implementors of that module. The anticipated result was that, when an expected change occurs, only one or two low-level module implementations (i.e., no interfaces) would need modification. Fig. 14 presents the distribution for the number of lowest-level modules updated by changes (i.e., the ripple effect of changes). Such modules were considered to be "updated" if their interface specifications (implementation documents, or code) were updated, unless the updates were to ancillary items such as indexes and tables of contents. Most early SCR changes (90%) updated zero or one modules, and this percentage is relatively constant. The data presented in Fig. 15 are a special case of the data presented in Fig. 14. Fig. 15 presents the distribution for the number of lowest-level modules which had interface specifications updated (i.e., interfaces updated because of changes). A module interface is considered to be "updated" if a change to its specification (or implementation document, or code) caused, or would have conceivably caused, a change to programs of other modules that use, or would eventually use, capabilities provided by the module. Examples of interface updates are the modification of a parameter type and the addition of a sysgen parameter. The percentage of early SCR changes that resulted in updated interface updates (56%) was growing. The percentage of changes updating two or more interfaces (12%) was also growing. These latter trends seem to suggest that a greater ripple effect and a more uniform distribution of change effort could have been expected later in the SCR project.

C. Change Data Related to Personnel Activity Data

SCR project engineers reported their activity weekly using activity forms designed by STE researchers (see Norcio and Chmura) [20]. The design and code data can be related to collected personnel activity data because origination activity was captured for each CRF (see Fig. 2). Fig. 16 presents the ratio of the cumulative changes uncovered during specific SCR activity (i.e., design, code, and test) to the cumulative project hours expended on that activity. Fig. 17 presents the ratio of cumulative hours for changes uncovered during an activity to the cumulative project hours expended on the activity. Interestingly, both show a similar pattern. Coding activity, which also includes pseudo coding activity, was the most "efficient" way for uncovering needed modifications and errors, followed closely by testing activity. But, this was true only initially. In the long run for the SCR project, it

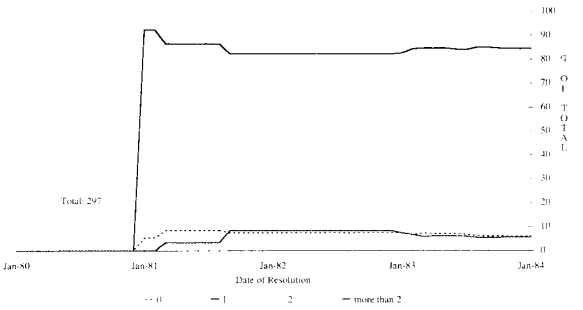


Fig. 14. Accepted CRF's categorized by number of modules updated.

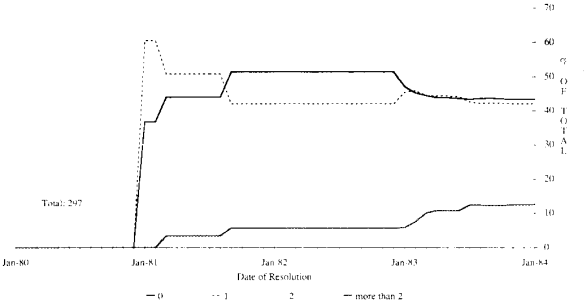


Fig. 15. Accepted CRF's categorized by number of interfaces updated.

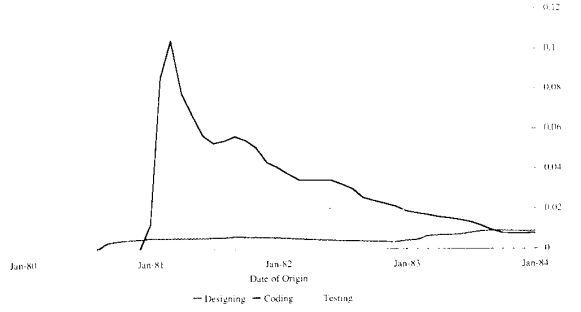


Fig. 16. Ratio of cumulative CRF's uncovered by an activity to cumulative project hours expended on activity.

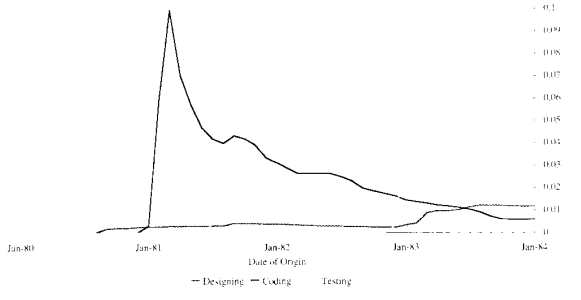


Fig. 17. Ratio of cumulative CRF resolution effort for CRF's uncovered by an activity to cumulative project hours expended on activity.

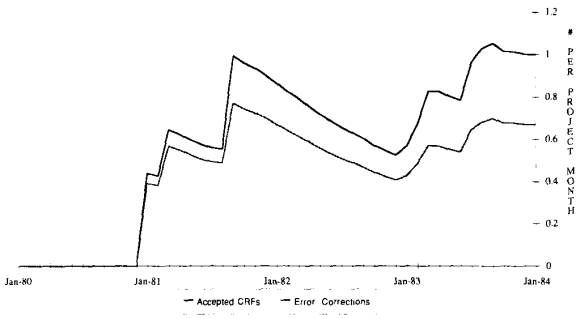


Fig. 18. Ratios of cumulative accepted CRF's and error corrections to cumulative project months.

seems that that design, code, and test activity were all equally efficient in terms of uncovering the need for changes. It should be noted, however, that the amount of coding (6504 hours) and testing (1188 hours) that accumulated by January 1984 were small compared to the amount of design (21 742 hours).

The ratio of cumulative error corrections to cumulative project work months and the ratio of cumulative accepted changes to cumulative project months appear in Fig. 18 (one work month equals 160 person hours). Although the ratios appear to be increasing, both are small compated to the data reported by Weiss and Basili [29]. They report approximately 2–3 error corrections per work month.

IV. Data Analyses

In previous analyses of SCR personnel activity data, Norcio and Chmura discovered that one ratio between two subactivities of SCR design activity correlates significantly over time with the cumulative design hours for the module [20]. The ratio is between a SCR module's cumulative *design discussing* hours and its cumulative *design creating* hours. The ratio has been referred to as the progress indicator ratio (PIR). When the release dates for module specification baselines are examined with respect to a graph of the PIR, patterns are readily apparent that may indicate relative instability of the module interface specification. SCR module interface specifications were rarely updated more than twice after this ratio became "stable" (i.e., showed small monthly change). In other words, if a specification baseline is issued before the ratio rises sharply or during a sharp rise, such a pattern seems to suggest that the baseline is probably far from complete.

A major complication with the PIR is that it requires a data collection scheme that accurately captures intricate information about personnel activity during the design process. Even though this seems possible to do accurately [7], it is fair to say that few software development efforts could readily afford and tolerate the collection operation. Because many design efforts routinely record software change data, we have looked at the SCR change data for information similar to that provided by the PIR. Fig. 16 suggests an alternative—a ratio between cumulative CRF's uncovered during design of a module and cumulative design hours for the module. It is an attractive alternative because intuition suggests that a module's interface design might be unstable while its designers are generating and resolving CRF's.

TABLE III
Second-Level Software Modules with Specification Baselines by January 1984

Abbreviation	Name
AT	Applications Data Type
DI	Device Interface
EC	Extended Computer
FD	Function Driver
SS	Shared Services

TABLE IV
Total Number of CRF's and Design Hours by December 1983

Module	CRFs Resulting From Design	Earliest CRF Date Of Origin	Design Hours
AT	2	Mar81	1084
DI	11	Sep80	2859
EC	119	Mar81	7478
FD	27	Sep80	1235
SS	6	Jan81	1848

Table III lists some of the second-level modules of the multilevel hierarchy of information-hiding modules resulting from the SCR design activity [5]. These modules had interface specifications with one or more baselines by January 1984. For each of the modules, two time-based ratios between the number of CRF's resulting from that module's design activity and the module's cumulative design hours can be computed and plotted. One ratio is based upon CRF date of origin; the other on date of resolution. Table IV is a summary of the data underlying these ratios for the modules listed in Table III.[4]

A. Date of Origin Ratio

For each module, the date of origin ratio (DOOR) is defined as the ratio of the cumulative CRF's by date of origin uncovered during design of the module to the cumulative design hours for the module. DOOR's for SCR modules are presented in Figs. 19–23. The vertical lines in these figures indicate isue dates for module specification baselines. Pearson product moment moment correlation coefficients (r) and coefficients of determination (r^2) between DOOR's and the original PIR's for each module with ten or more CRF's are presented in Table V [14]. The time period over which correlations are computed begins with the date of origin of the earliest CRF as presented in Table IV.

As can be seen in Table V, the correlation between DOOR and PIR for FD module is negative. This is not a problem. It merely means the two ratios are slightly oscillating in opposite directions. The important and significant point is that (r^2) is necessarily positive and signifi-

[4]Even though the number of CRFs for the AT module is only 2, these data are reported here and in Figs. 14 and 24 for completeness. These data for this module were not used in the subsequent statistical analyses.

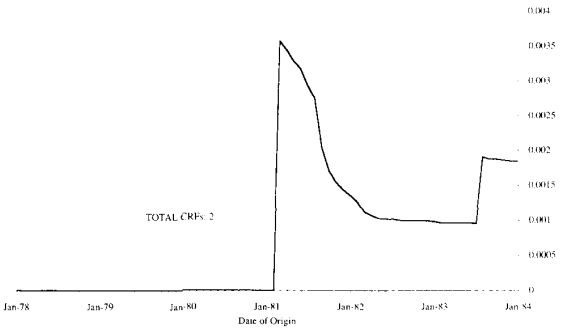


Fig. 19. Date of origin ratio for AT.

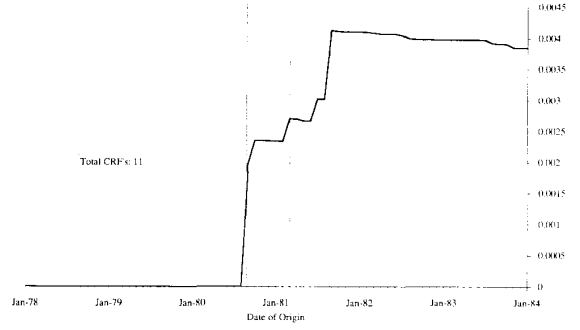


Fig. 20. Date of origin ratio for DI.

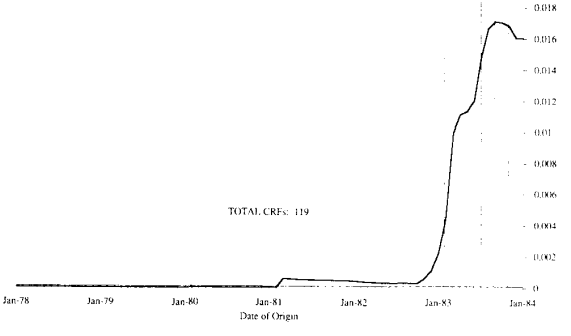


Fig. 21. Date of origin ratio for EC.

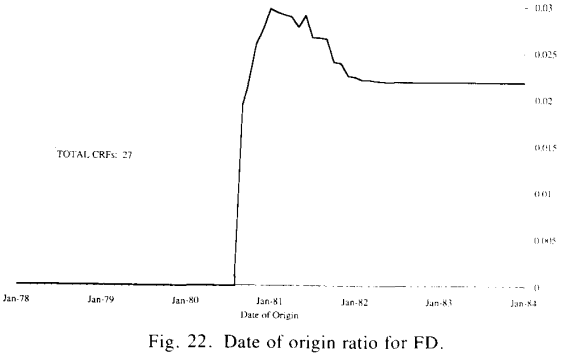


Fig. 22. Date of origin ratio for FD.

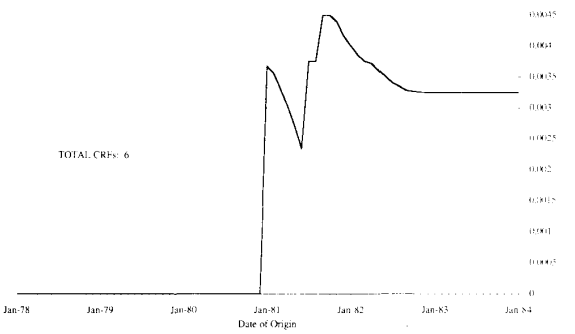


Fig. 23. Date of origin ratio for SS.

TABLE V
PEARSON CORRELATION COEFFICIENTS BETWEEN DOOR AND PIR

Module	r	r^2
DI	0.727	0.528
EC	0.985	0.970
FD	−0.679	0.461

cantly high, which means that both ratios are behaving in very similar fashions.

B. Date of Resolution Ratio

The date of resolution (DORR) is the same as the DOOR except that CRF date of resolution is used rather than date of origin. DORR's for SCR modules are presented in Figs. 24–28. Again, vertical lines indicate baseline issue dates. Pearson product moment correlation coefficients (r) and coefficients of determination (r^2) between DORR's and the original PIR's for each module with ten or more CRF's are presented in Table VI [14]. The time period over which correlations are computed is the same as for the DOOR.

C. Possible Implications

Analyses of the design CRF data suggest that, in some cases, fairly simple change and personnel activity data may be used as an alternative to the originally proposed PIR. The DOOR's and the DORR's for modules with a significant number of design changes show a strong relationship to the original PIR's. The DOOR explains 52, 97 and 46% of the variation in the original PIR's for the DI, EC, and FD modules; the DORPIR, 49, 94, and 50%.

When issue dates for published baselines are superimposed upon the DOOR and DORR plots, patterns reminiscent of those observed with the original PIR are observed. Baselines that appear during times of instability in the DOOR or DORR are soon followed by other baselines. For module designs that have been specified with only one or two baselines, one sees a prior instability with the DOOR and DORR, a downward trend, issuance of the baseline, and then relative stability. For other modules, this pattern is lacking for one or more of the earlier base-

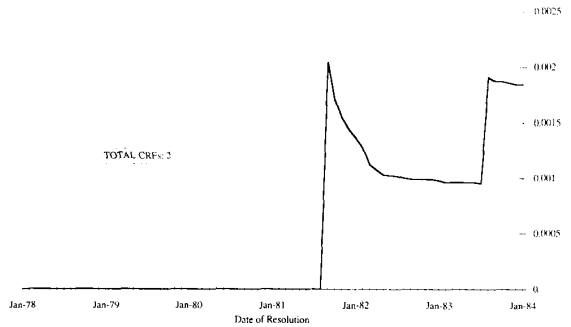


Fig. 24. Date of resolution ratio for AT.

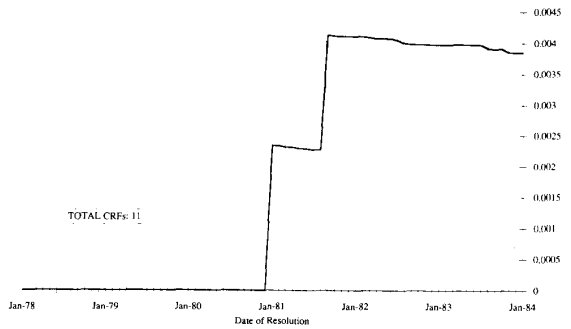


Fig. 25. Date of resolution ratio for DI.

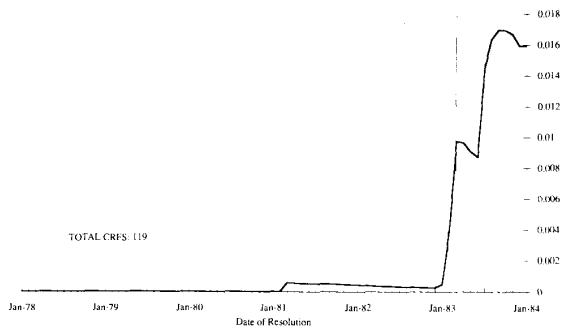


Fig. 26. Date of resolution ratio for EC.

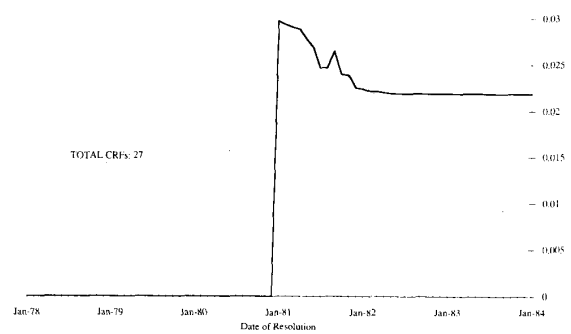


Fig. 27. Date of resolution ratio for FD.

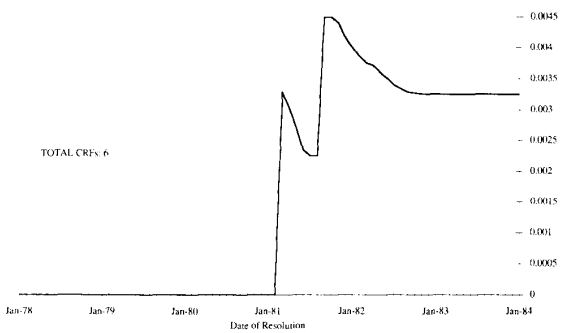


Fig. 28. Date of resolution ratio for SS.

TABLE VI
PEARSON CORRELATION COEFFICIENTS BETWEEN DORR AND PIR

Module	r	r^2
DI	0.698	0.487
EC	0.971	0.943
FD	0.709	0.503

lines. In other words, the DOOR and DORR both may indicate the incompleteness of interface specifications. If these ratios have not surged and then turned downwards prior to appearance of a baseline and, subsequently stabilized, then the design of the module's interface may not be complete, irrespective of the claims of software engineers and the information in published documents.

V. SUMMARY AND CONCLUSIONS

A study of the SCR project's early change data and analyses of time-based relationships shows the following.

1) There was a high proportion of error corrections and error correction effort, although time-based plots of these statistics show that both were on the decrease.

2) The percentage of error corrections that involved completing or correcting a prior change was far higher than has ever been reported, and this percentage was increasing.

3) The percentage of changes that took a day or less to resolve was extremely large, but was decreasing. Consistent with this decrease was a stepwise growth in average change effort, a growth in the percentage of changes that involve modifying module interfaces, and a growth in the percentage of changes involving two or more module interfaces.

4) Surprisingly, no relationship was shown between change effort and number of days that an error exists in the documentation.

5) Coding activity, followed by testing activity, was the most efficient way of uncovering needed modifications and error corrections. In the long run, however, it seems

that design, code, and test activity were all equally efficient.

Analyses of the design CRF data and their relationships to personnel activity data show two ratios that may be useful to design managers in assessing the progess of the software design process. Referred to as the DOOR and DORR, the ratios exhibit patterns seemingly related to the incompleteness of interface specifications. If these ratios have not surged and then turned downwards prior to the appearance of a baseline and, subsequently stabilized, then it would not be surprising to see several more specification baselines in the future. The ratios are attractive alternatives to an earlier-reported PIR ratio because they are based on simple design activity data and on change data close to the kinds typically collected on software projects.

There are some drawbacks to the DOOR and the DORR as potential indicators of design progress. One is that they are later indicators as compared to the original PIR. Another is that they are based heavily on the responsiveness and timeliness of a project's change control process. If changes are not resolved promptly, any potential relationships between these ratios and design progress may be weakened.

It must be noted that we do not claim that the DOOR or DORR are measures of the completeness of an interface design. There may be many reasons why the ratios stabilize (e.g., personnel have been assigned to another module or have taken vacations). For SCR modules, however, the ratios do show readily apparent patterns that are strikingly different for modules with a history of many specification baselines than for those without such a history.

Acknowledgment

We are especially indebted to P. Clements who, as lead SCR software engineer, patiently assisted CRF validators in resolving problems that were encountered. Another contributor was Ms. K. Kragh, who for several years entered change and activity data into a computer database, checked the accuracy of each entry, and updated everything when, for example, the names of modules changed. Without her diligence, this paper could not have been written. We would also like to thank the referees and editor for thoughtful and helpful suggestions on earlier drafts of this paper.

References

[1] V. R. Basili and B. T. Perricone, "Software errors and complexity: An empirical investigation," *Commun. ACM*, vol. 27, pp. 42-52, 1984.

[2] V. R. Basili and D. M. Weiss, "A methodology for collecting valid software engineering data," *IEEE Trans. Software Eng.*, vol. SE-10, no. 6, pp. 728-738, 1984.

[3] L. A. Belady and M. M. Lehman, "A model of large program development," *IBM Syst. J.*, vol. 3, pp. 225-252, 1976.

[4] B. W. Boehm, *Software Engineering Economics*. Englewood Cliffs, NJ: Prentice-Hall, 1981.

[5] K. H. Britton and D. L. Parnas, A-7E Module Guide, Naval Res. Lab., Washington, DC, Rep. 4702, 1981.

[6] L. J. Chmura, "Proposed new design and code change report form (CRF) for the software cost reduction (SCR) project," Naval Res. Lab., Washington, DC, Tech. Memo. 7590-34:LC, 1983.

[7] L. J. Chmura and A. F. Norcio, "Accuracy of software activity data: The software cost reduction project," Naval Res. Lab., Washington, DC, Rep. 8780, 1983.

[8] L. J. Chmura and D. M. Weiss, "The A-7E software requirements document: Three years of change data," Naval Res. Lab., Washington, DC, Memo. Rep. 4938, 1982.

[9] V. Church, D. Card, F. McGarry, and V. Basili, "Guide to data collection," NASA/Goddard SEL, Greenbelt, MD, Tech. Rep. SEL-81-001, 1981.

[10] P. C. Clements, "Software cost reduction through disciplined design," *Naval Res. Lab. 1984 Rev.*, pp. 79-87, 1985.

[11] P. C. Clements, R. A. Parker, D. L. Parnas, and J. Shore, "A standard organization for specifying abstract interfaces," Naval Res. Lab., Washington, DC, Rep. 8815, 1984.

[12] R. Day, "A history of software maintenance for a complex U.S. Army battlefield automated system," in *Proc. Conf. Software Maintenance*, 1985, pp. 181-187.

[13] E. W. Dijkstra, "Cooperating sequential processes," in *Programming Languages*, F. Genuys, Ed. New York: Academic, 1968, pp. 43-112.

[14] W. J. Dixon and F. J. Massey, Jr., *Introduction to Statistical Analysis*. New York: McGraw-Hill, 1969.

[15] A. Endres, "An analysis of errors and their causes in system programs," *IEEE Trans. Software Eng.*, vol. SE-1, no. 2, pp. 140-149, 1975.

[16] R. B. Grady, "Measuring and managing software maintenance," *IEEE Software*, pp. 35-45 Sept. 1987.

[17] K. L. Heninger, J. W. Kallander, J. E. Shore, D. L. Parnas, and Staff, "Software requirements for the A-7E aircraft," Naval Res. Lab., Washington, DC, Memo. Rep. 3876, 1978.

[18] B. P. Lientz, E. B. Swanson, and G. E. Tompkins, "Characteristics of application software maintenance," *Commun. ACM*, vol. 21, no. 6, pp. 466-471, 1978.

[19] B. Meyer, "On formalism in specifications," *IEEE Software*, vol. 2, no. 1, pp. 6-27, 1985.

[20] A. F. Norcio and L. J. Chmura, "Design activity in developing modules for complex software," in *Empirical Studies of Programmers*, E. Soloway and S. Iyengar, Eds. Norwood, NJ: Ablex, 1986.

[21] A. Parker, K. L. Heninger, D. Parnas, and J. Shore, "Abstract interface specification for the device interface module," Naval Res. Lab., Washington, DC, Memo. Rep. 4385, 1980.

[22] D. L. Parnas, "On the criteria to be used in decomposing systems into modules," *Commun. ACM*, vol. 15, pp. 1053-1058, 1972.

[23] ——, "Use of abstract interfaces in the development of software for embedded systems," Naval Res. Lab., Washington, DC, Rep. 8047, 1977.

[24] D. L. Parnas and P. C. Clements, "A rational design process: How and why to fake it," *IEEE Trans. Software Eng.*, vol. SE-12, no. 2, pp. 251-257, 1986.

[25] D. L. Parnas, P. C. Clements, and D. M. Weiss, "The modular structure of complex systems," *IEEE Trans. Software Eng.*, vol. SE-11, no. 3, pp. 259-266, 1985.

[26] M. L. Shooman and M. I. Bolsky, "Types, distribution, and test and correction times for programming errors," in *Proc. Int. Conf. Reliable Software*, pp. 347-357.

[27] E. B. Swanson, "The dimensions of maintenance," in *Proc. Second Int. Conf. Software Engineering*, 1976, pp. 492-497.

[28] D. M. Weiss, "A comparison of errors in different software-development environments," Naval Res. Lab., Washington, DC, NRL Rep. 8598, 1982.

[29] D. M. Weiss and V. R. Basili, "Evaluating software development by analysis of changes: Some data from the Software Engineering Laboratory," *IEEE Trans. Software Eng.*, vol. SE-11, no. 2, pp. 157-168, 1985.

[30] R. Z. Xu, "An empirical investigation: Software errors and their influence upon development of WPADT," in *Proc. COMPSAC 85*, 1985, pp. 4-8.

L. J. Chmura, Jr. received the Bachelor's degree in mathematics and the Master's degree in computer science.

He is a Computer Scientist with the Terrestrial Systems Branch of the Naval Research Laboratory in Washington, DC. His research interests include software development methodologies, development environments, empirical studies of methodologies, and human-computer interfaces.

A. F. Norcio received the Bachelor's degree in economics, the B.S. degree in statistics, and the Ph.D. degree in psychology.

He is an Associate Professor in the Department of Information Systems Management of the University of Maryland, Baltimore County. He is also a Computer Scientist at the Human-Computer Interaction Laboratory of the Naval Research Laboratory in Washington, DC. His research interests are in the areas of human-computer interfaces, software design, and intelligent systems. He has published several dozen papers in a variety of journals and periodicals. He has served on planning committees for many major national and international conferences, and regularly reviews papers for journals of several learned societies and professional publishers.

Dr. Norcio is a member of the American Association for Artificial Intelligence, the Association for Computing Machinery, the Human Factors Society, and the IEEE Computer Society.

T. J. Wicinski, photograph and biography not available at the time of publication.