PROCEEDINGS
OF THE
TWELFTH ANNUAL SOFTWARE ENGINEERING WORKSHOP

Organized by:
Software Engineering Laboratory
GSFC

December 2, 1987

GODDARD SPACE FLIGHT CENTER
Greenbelt, Maryland
Analyses of Software Design Changes

L. J. Chmura and A. F. Norcio and T. J. Wicinski

Computer Science and Systems Branch
Information Technology Division
Naval Research Laboratory
Washington, D. C. 20375-5000

November 30, 1987
Abstract

This paper presents analyses of early design and code change report forms (CRFs) from the Software Cost Reduction (SCR) project at the Naval Research Laboratory. Using software engineering principles such as information hiding, SCR engineers are redesigning the operational flight program for the Navy's A-7E aircraft. The two major goals are to demonstrate the effectiveness of such techniques in developing, for example, easy to change hard-real-time software and to provide high-quality development models for others to follow.

The first part of this paper is a characterization of the SCR development effort and an examination of how well it is meeting its goals based on analyses of CRF data. Results are presented as time-based trends are compared to similar data reported for other projects. One characteristic of the SCR project seems to be that it exhibits a comparatively high percentage of changes involving error corrections (67%) and of change effort devoted to error corrections (50%). However, both of these percentages are on the decline. Another characteristic is that the percentage of errors that are corrections or completions of prior changes (28%) is quite large and seems to be increasing. Consistent with the project's goal for easy-to-change software, almost all changes (98%) took a day or less to uncover and resolve, though this percentage is dropping. Also, most changes (90%) updated at most one module, and this percentage seems stable. But, there are some ominous trends. There is a stepwise growth in average change effort; and, the percentage of changes that result in module interface updates (58%) is growing. Surprisingly, however, there does not seem to be a relationship between error correction effort and the number of days that an error exists in the SCR design documentation. Equally surprising is that SCR coding activity, followed by testing activity, are the more efficient (uncovered changes per person hour) methods for uncovering needed changes and errors only initially. In the long run,
design, code, and test activity appear equally efficient.

The second part of this paper is an examination of time-based ratios between SCR change data and personnel activity data and the possible general use of the ratios as indicators of design progress. Two similar ratios are identified that show promise as indicators of design incompleteness. For an information hiding module, these are ratios between the cumulative CRFs uncovered during the design of the module and the cumulative design hours for the module. One ratio is based on CRF date of origin while the other is based on date of resolution. Both ratios are intuitively appealing and offer a simpler alternative to an earlier reported progress indicator ratio (PIR).
I. Introduction

This paper presents analyses of early design changes proposed and made by software development engineers working on the Software Cost Reduction project at the Naval Research Laboratory (NRL). There are five sections in the paper. The remainder of this section contains an overview of NRL's Software Cost Reduction 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 next two sections contain the analyses of the data and their implications.

The Software Cost Reduction Project

Since 1978, the Naval Research Laboratory, in cooperation with the Naval Weapons Center, has been redeveloping version 2 of the operational flight program for the A-7E aircraft [9]. Software engineering techniques such as formal requirements specification, information hiding [19], abstract interfaces [20], and cooperating sequential processes [11] are being used. This research effort is referred to as the Software Cost Reduction (SCR) project.

The goals of the project are to (1) demonstrate the feasibility of using selected software engineering techniques in developing complex, real-time software, and (2) provide a model for software design. The claimed advantage of the selected software engineering techniques is that they facilitate the development of software that is easy to change and maintain. A complete discussion of the project's software requirements is provided by Heninger, et al [14]. For a detailed description of the module design structure, the reader is referred to Britton and Parnas [4]. Figure 1 presents an example of a module interface specification (i.e., a design specification) taken from a recent version of the specification for the device interface module [18]. A standard organization for such specifications is described by Clements et al [8].
The SCR project terminates at the end of 1987 after producing three successive versions that implemented increasingly larger subsets of the operational flight program requirements. The subsets are being evaluated and tested using ground-based test facilities.

The Software Technology Evaluation Project

The Software Technology Evaluation (STE) project is a separate research project from the SCR project in terms of goals, staffing, and funding. The goal of the STE project is to evaluate alternative software development technologies. A major task of the STE project, therefore, is to provide the basis for an objective evaluation of the methodology used in the SCR project.

The approach followed in the STE project is to monitor, evaluate, and compare software development technologies used in different software projects. The monitoring and evaluating processes consist of goal-directed data collection and analyses techniques [2]. For the SCR project, data has been collected in three areas: personnel activity [16], changes to requirements [7], and changes to design and code. This paper is the first published analysis of SCR design and code change data.

II. Collection Of Change Data

From 1980 until early 1985, SCR project engineers reported design and code problems as well as suggested design changes. Further, they logged their modification activity to baselined (i.e., published and change-controlled) interface specifications, pseudo code, and TC-2 code2 on Change Report Forms (CRFs). An example of a completed CRF is presented in Figure 2. There

---

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 model TC-2 computer. The operational flight program runs on this machine.
are two reasons for this procedure. First, it is required as part of SCR project configuration management (CM) procedures. Second, such data are needed by STE researchers for evaluating achievement of SCR project goals. The specific design of the CRF form is based on a goal-directed data collection approach [6]. In 1985, the use of paper CRFs was discontinued. Since then, SCR engineers have noted problems and proposed changes using a computer-based configuration management tool.

STE researchers have validated primarily those CRFs that have been resolved either by official acceptance and incorporation into the design, or by official rejection of the proposed change. Ideally, validation should be a continuing activity that occurs as CRFs are generated and resolved. Validation of SCR CRFs, however, has tended to be an aperiodic activity in which large groups of CRFs are validated at one time. The validation consists of checking completeness, accuracy, etc. It often includes discussions with persons who submitted the CRFs, authors of affected documents, and SCR CM personnel. A major validation point concerns what constitutes a design or code change. Basically, the view taken is that a change is conceptual; that is, one should be able to state a proposed change a simple declarative sentence and may comprise alterations to one or more baselined interface specifications or implementation documents. In addition, a change must have a unique basis — error correction, adaptation to outside change, improvement, or other (see Figure 2). Thus, a change that is described in one CRF similar to a change in a CRF resolved and implemented in earlier baselines (i.e., a change that requires completion or correction to earlier baselined alterations) is a unique or new change. A proposed change that is rejected obviously results in no alterations.

This definition of a design or code change can cause problems. Occasionally a CRF is submitted that incorporates more than one change, and different engineers sometimes submit the same change on different CRFs. For example, it is not unusual for a CRF to describe two con-
ceptual changes.

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

The STE project's policy for dealing with these situations is to split submitted CRFs that incorporate more than one change into an appropriate number of CRFs, such that each describes a single change. Multiple CRFs that describe identical changes are consolidated into one CRF. One result of this policy is that there is not a one-to-one correspondence between submitted CRFs and validated CRFs. The other result is, of course, that there is a one-to-one correspondence between proposed changes and validated CRFs.

There are other sections of the CRF that cause difficulties. One is determining the basis of an accepted change. The CRF classification scheme appearing at the bottom of the front side of the change form follows from the scheme presented by Swanson [22]. One problem is that it is not sufficient to define an error as a discrepancy between a specification and its implementation. For example, it is sometimes difficult to decide if a CRF describes an inadequate interface design (i.e., an error) or if it simply describes a better design (i.e., an improvement). The policy has been to let SCR lead engineers decide between these situations. A second problem is determining whether or not a change is a correction or completion of an earlier change that has already been baselined. The fact is, 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 CRFs. For each of the CRFs reported in this study, STE researchers have reviewed all versions of all documents baselined prior to resolution of the CRF and discussed all questions with lead SCR engineers. This was done to ensure that corrections or completion errors are properly identified.
Lastly, the SCR project's CM procedures are not perfect. Validators have found CRFs that have not been resolved, but, nevertheless, are implemented in published specifications. The policy for this has been to resolve such CRFs with the date of the latest issued baseline specification and to submit CRFs for remaining aspects of the change. Validators have also found modifications for which there were no corresponding CRFs. The policy for this has been to submit CRFs and have them immediately resolved with the date of issue of the baseline specification.

III. Overview of Early SCR Change Data

General

This study reports on 325 validated CRFs that were resolved by January 1984 (i.e., through the end of December, 1983). By January 1984, engineers had submitted 424 CRFs. The 325 validated CRFs reported here map to 296 (70%) of those submitted that were resolved by SCR CM personnel by that date. Figures 3 and 4 are profiles of resolution activity for these proposed changes. 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 1 presents the distribution of the CRFs categorized by the originators' activities when the CRFs were generated. In addition, only 15% of SCR project hours were spent on pseudo coding, coding, and testing activities by January 1984. Thus, the changes reviewed here can be characterized as changes that are typically proposed and made early in software development, which contrasts with changes reported elsewhere [1,13,23].

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 Figures 5 and 6). The 9% figure is
Table 1. Activities Leading to CRF Origination

<table>
<thead>
<tr>
<th>Project Activity:</th>
<th>Count (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Design (e.g., Module Interface Specification)</td>
<td>209 (64%)</td>
</tr>
<tr>
<td>Pseudo Code</td>
<td>53 (16%)</td>
</tr>
<tr>
<td>Code</td>
<td>1 (0%)</td>
</tr>
<tr>
<td>Test</td>
<td>26 (8%)</td>
</tr>
<tr>
<td>Misc</td>
<td>15 (5%)</td>
</tr>
<tr>
<td>Unknown</td>
<td>5 (2%)</td>
</tr>
<tr>
<td>**Total:</td>
<td><strong>309 (95%)</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Non-Project Activity (e.g., CRF validation):</th>
<th>Count (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Total:</strong></td>
<td><strong>16 (5%)</strong></td>
</tr>
</tbody>
</table>

Small compared to both the 37% figure reported by Day [10] for major maintenance updates to an operational Army command and control system and the 20% figure reported by Shooman and Bolsky [21] for errors discovered and corrected during test and integration of a modest-size control program at Bell Telephone Laboratories. The 4% effort figure is comparable to the 3% figure reported by Day. Care must be taken with these comparisons, however. These two figures are from different times in different project life cycles, and it is not clear if there is a common definition of change. More importantly, SCR requirements changes are a separate SCR CM concern and are not incorporated in the data reported here [7].

The remaining 297 accepted CRFs resulted in modifications to baselined items. The bases for these changes are presented in Table 2. None of the changes were the result of changes to the software requirements specification. This can probably be attributed to the following:

1. The SCR project is redeveloping software for a fixed operational version of the A-7E flight software,

2. A rather extensive requirements specification was generated prior to design [14],

3. The requirements specification has been shown to be relatively error free and remarkably free of ambiguities [7], and
(4) as noted earlier, the changes reported are early changes.

Actually, all 297 changes required updates to only 47 baselined module interface specifications, most of which are packaged in two documents. The primary reason for this is that no module implementation documents (which include pseudo code) were baselined prior to January 1984. In other words, the 297 changes can be considered to be early design changes.

The percentage of error corrections (see Table 2 and Figure 7) is high compared to data reported for other development efforts [1,21,23], but is decreasing. The proportion of total CRF effort spent on error corrections (Figure 8) sharply contrasts with the 17% figure reported by Lientz and Swanson [15] for commercial data processing software maintenance efforts, and the 21% figure reported by Day [10], but, this percentage is also decreasing. It should be noted again however, that 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 Figure 9) is large as compared to the 6% - 12% range of figures reported by others [1,24,23] and seems to be increasing in a step fashion. The 12% figure is computed from data presented by Weiss [24] and by Weiss and Basili [23].

Table 2. Bases Of Accepted CRFs

<table>
<thead>
<tr>
<th>Error Corrections:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Original</td>
<td>144 (48%)</td>
</tr>
<tr>
<td>Continuation or Completion</td>
<td>55 (19%)</td>
</tr>
<tr>
<td><strong>Total: 199 (67%)</strong></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Modifications:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Adaptation to requirements change</td>
<td>0 (0%)</td>
</tr>
<tr>
<td>Adaptation to support environment change</td>
<td>0 (0%)</td>
</tr>
<tr>
<td>Improvement in performance</td>
<td>2 (1%)</td>
</tr>
<tr>
<td>Improvement in clarity</td>
<td>89 (30%)</td>
</tr>
<tr>
<td>Other</td>
<td>7 (2%)</td>
</tr>
<tr>
<td><strong>Total: 98 (33%)</strong></td>
<td></td>
</tr>
</tbody>
</table>
Ease of Change

A major objective of the SCR project is to produce a software design, code, and a documentation set that can be used to scope and implement changes easily. Figure 10 presents the distribution of effort required for understanding and incorporating the 297 accepted changes into the SCR project’s design and documentation set; Figure 11 presents the distribution for error corrections only. Only one of the 28 rejected CRFs was not implemented because the proposed change was deemed not worth the effort. Many 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. Though exhibiting downward trends, these figures seem to suggest that, for early changes and error corrections, SCR engineers are meeting their major objective. For errors uncovered and corrected late in the life cycle of a NASA/Goddard Software Engineering Laboratory project, Basili and Perricone [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 WPADT project, Xu [26] reports 24% of the error corrections project took an hour or less; and 80% took a day or less.

Figure 12 presents the cumulative average effort for all SCR changes and error corrections. There seems to be a stepwise growth in cumulative average change effort as the SCR project life cycle lengthens. This is consistent with Boehm’s [3] data that shows an exponential growth in cost to fix or change software for successive phases of the software life cycle. In terms of this result, the SCR project seems no different than other software development projects. Figure 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 considered to be 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 .07. There may be two reasons for this. The first is that SCR requirements change data are not included 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 to pass before any relationship appears.

The information hiding principle is used in the SCR project for identifying and specifying modules. A module is supposed to hide a likely changeable aspect of the A-7E flight software. This means that a module's interface specification must be written such that the hidden information is not revealed; that is, a module's hidden information is available only to the implementors of that module. The anticipated result is that, when an expected change occurs, only one module implementation (i.e., no interface) needs modification. Figure 14 presents the distribution for the number of modules updated by changes (i.e., the ripple effect of changes). A module is updated if its interface specification (implementation document, or code) is updated. A change is considered to update zero modules if updates are required in other documentation or in indexes and tables of contents associated with packaged sets of module specifications. Most changes (90%) updated zero or one modules, and this percentage is relatively constant. Figure 15 presents the proportion of changes that resulted in module interface updates. A module interface is updated if a change to its specification (or implementation document, or code) causes 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 modification of a parameter type or addition in a sysgen parameter. The percentage of changes that resulted in updated interface updates (56%) is growing. The percentage of changes updating two or more interfaces (12%) is also growing. These latter trends seem to suggest that a greater ripple effect and a more uniform distribution of change effort can be expected in the future.
Error Causes

In Figure 16, the distribution of error causes is presented. Thirty-three percent of the error corrections are clerical in nature; that is, they are characterized as likely to have been made when an engineer or secretary was typing the material. This percentage, which is growing, is large in comparison to other reported data (e.g. [1]); Weiss [25], however, has reported a 36% figure for an earlier NRL software project, the Architecture Research Facility.

The majority of errors (65%) have "other" causes. An examination of these causes shows that engineers attributed the errors to failings on their part. Thus, this percentage is close to the 68% figure for programmer error reported by Ostrand and Weyuker [17].

Only 2 errors (1%) were felt to be caused by poor SCR documentation! This contrasts to the 9% figure for poor documentation reported by Ostrand and Weyuker. Either SCR engineers are reluctant to fault their documentation or their documentation is quite good or they simply tend to blame themselves for errors. The last 1% of errors had unknown causes.

Change Data Related to Personnel Activity Data

SCR project engineers report their activity weekly to their project management using forms designed by STE researchers. Figure 17 presents the ratio of the cumulative changes uncovered during a specific activity (i.e., design, code, and test) to the cumulative hours that were expended in that activity. Figure 18 presents the ratio of cumulative hours for changes uncovered during an activity to the cumulative activity hours. Both show a similar pattern. Coding activity, followed closely by testing activity, is the most "efficient" way for uncovering needed modifications and errors, but, only initially. In the long run, for the SCR project, it seems that that design, code, and test activity are all equally efficient in terms of uncovering changes. However, the amount of coding (6504.25 hours) and testing (1487.5 hours) that accu-
mulated by January 1984 is small compared to the amount of design (21741.75 hours).

The proportion of error corrections for a project work month and the proportion of implemented changes for a work month (i.e., 160 person hours) appear in Figure 19. Although they appear to be increasing, both ratios are small compared to the data reported by Weiss and Basili [23]. They report approximately 2 – 3 error corrections per work month and 4 – 8 changes per work month.

IV. Data Analyses

In a previous study of SCR Project activity data, Norcio and Chmura have defined the Progress Indicator Ratio (PIR) [16]. The PIR, which is a time-based ratio between a module's cumulative design discussing hours and cumulative design creating hours, correlates consistently well with total design hours for the module. When the release dates for specification baselines are examined in conjunction with the PIR, the PIR seems to indicate incompleteness of baseline specifications. The appearance of a baseline before the PIR rises sharply or during a sharp rise seems to suggest that the baseline is probably far from complete. Module interface specifications seem to become reasonably stable only when the PIR becomes stable.

A major objection to 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 [5], 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, it would be desirable if information provided by the PIR could also be provided by change data. Figure 17 suggests a possible use of change data. Also, one's intuition suggests that a module's interface design would be unstable if people who were working on that design were generating many CRFs against the current version of the design or against the
interface designs of other modules.

Table 3 lists some of the second-level modules of the multilevel hierarchy of information-hiding modules resulting from the SCR design activity [4]. These modules have interface specifications that have had one or more baselines, and each have been modified by one or more of the 325 CRFs. For each of the modules, time-based ratios between the number of CRFs resulting from module design activity and the cumulative module design hours can be computed and plotted. These are the Date of Origin PIR (DOOPIR) and the Date of Resolution PIR (DORPIR), based upon CRF date of origin and resolution respectively. Table 4 is a summary of the data underlying these ratios; specifically, they are the number of CRFs that resulted during design work on the module, the date of origin of the earliest of those CRFs, and the total design hours for each module.

Table 3. Abbreviations and Names of Second Level Software Modules

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>AT</td>
<td>Applications Data Type</td>
</tr>
<tr>
<td>DI</td>
<td>Device Interface</td>
</tr>
<tr>
<td>EC</td>
<td>Extended Computer</td>
</tr>
<tr>
<td>FD</td>
<td>Function Driver</td>
</tr>
<tr>
<td>SS</td>
<td>Shared Services</td>
</tr>
</tbody>
</table>
Table 4. Total Number of CRFs and Design Hours Through December 1983

<table>
<thead>
<tr>
<th>Module</th>
<th>CRFs Resulting From Design</th>
<th>Earliest CRF Date Of Origin</th>
<th>Design Hours</th>
</tr>
</thead>
<tbody>
<tr>
<td>AT</td>
<td>2</td>
<td>Mar81</td>
<td>1083.75</td>
</tr>
<tr>
<td>DI</td>
<td>11</td>
<td>Sep80</td>
<td>2859.00</td>
</tr>
<tr>
<td>EC</td>
<td>119</td>
<td>Mar81</td>
<td>7477.50</td>
</tr>
<tr>
<td>FD</td>
<td>27</td>
<td>Sep80</td>
<td>1235.05</td>
</tr>
<tr>
<td>SS</td>
<td>6</td>
<td>Jan81</td>
<td>1848.45</td>
</tr>
</tbody>
</table>

Date of Origin PIR

For each module, the DOOPIR is defined as the ratio between the cumulative CRFs by date of origin uncovered during design of the module and cumulative design hours for the module. DOOPIRs for each module are presented in Figures 20 to 24. The vertical lines in these figures indicate issue dates for module specification baselines. Pearson product moment correlation coefficients \( r \) and coefficients of determination \( r^2 \) between DOOPIRs and the original PIRs for each module [12] are presented in Table 5. The time period over which correlations are computed begins with the date of origin of the earliest CRF as presented in Table 4.
Table 5. Pearson Correlations Coefficients between DOOPIR and PIR

<table>
<thead>
<tr>
<th>Module</th>
<th>$r$</th>
<th>$r^2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>AT</td>
<td>-0.610*</td>
<td>0.372*</td>
</tr>
<tr>
<td>DI</td>
<td>0.727</td>
<td>0.528</td>
</tr>
<tr>
<td>EC</td>
<td>0.985</td>
<td>0.970</td>
</tr>
<tr>
<td>FD</td>
<td>-0.679</td>
<td>0.461</td>
</tr>
<tr>
<td>SS</td>
<td>-0.478*</td>
<td>0.228*</td>
</tr>
</tbody>
</table>

*Not significant at the $p \geq .05$ level.

*Date of Resolution PIR*

The DORPIR is the same as the DOOPIR except that CRF date of resolution is used rather than date of origin. DORPIRs for each module are presented in Figures 25 to 29. Again, vertical lines indicate baseline issue dates. Pearson product moment correlation coefficients ($r$) and coefficients of determination ($r^2$) between DORPIRs and the original PIRs for each module [12] are presented in Table 6. The time period over which correlations are computed is the same as for the DOOPIR. Even though the date of resolution occurred after the date of origin, hours of resolution effort include origination time plus subsequent change time.
Table 6. Pearson Correlations Coefficients between DORPIR and PIR

<table>
<thead>
<tr>
<th>Module</th>
<th>r</th>
<th>r^2</th>
</tr>
</thead>
<tbody>
<tr>
<td>AT</td>
<td>0.391*</td>
<td>0.152*</td>
</tr>
<tr>
<td>DI</td>
<td>0.698</td>
<td>0.487</td>
</tr>
<tr>
<td>EC</td>
<td>0.971</td>
<td>0.943</td>
</tr>
<tr>
<td>FD</td>
<td>0.709</td>
<td>0.503</td>
</tr>
<tr>
<td>SS</td>
<td>-0.472*</td>
<td>0.223*</td>
</tr>
</tbody>
</table>

*Not significant at the p ≥ 0.05 level.

V. Results and Conclusions

An overview of the SCR project’s early change data with respect to customary concerns and a time-based view shows the following major patterns.

1. There is a high proportion of error corrections and error correction effort, though time-based plots of these statistics show that both are on the decrease.

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

3. The percentage of changes that took a day or less to resolve is extremely large, but is decreasing. Consistent with this decrease is 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 has been shown between change effort and number of days that an error exists in the documentation.
5. Very few errors have been attributed to poor project documentation.

6. Coding activity, followed by testing activity, is the most efficient way of uncovering needed modifications and error corrections. In the long run, however, it seems that design, code, and test activity are all equally efficient.

An analyses of the design CRF data has suggested that, in some cases, fairly simple change and personnel activity data can be used as an alternative to the originally proposed PIR. First, the DOOPIRs and the DORPIRs for modules with a significant number of design changes show a strong relationship to the original PIRs. It appears that 10 CRFs can be considered a reasonable threshold for sensitivity. The DOOPIR explains 52%, 97%, and 46% of the variation in the original PIRs for the DI, EC, and FD modules; the DORPIR, 49%, 94%, and 50%.

Second, when issue dates for published baselines are superimposed upon the DOOPIR and DORPIR plots, patterns reminiscent if not even more sensitive than those observed with the original PIR are seen. For module designs that have been specified with only one or two baselines, one sees a prior instability with the DOOPIR and DORPIR, a downwards trend, issuance of the baseline, and then relative stability. For other modules, this pattern is lacking for one or more of the earlier baselines. In other words, the DOOPIR and DORPIR both seem to be indicating 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 probably is not complete irrespective of personnel claims and published documents.

There are two drawbacks to the DOOPIR and the DORPIR. The first is that they are later indicators of design progress than the original PIR. The second is that they are based heavily on the responsiveness and timeliness of a project’s change control process. If changes are not resolved promptly, the relationships between these ratios and design progress are weakened.

L. Chmura
Naval Research Lab
19 of 63
Finally, there is no claim that the DOOPIR or DORPIR 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). However, the ratios do seem to indicate when work on an interface is not complete. If completion is claimed prior to a downward trend and subsequent stability, there is probably more work that needs to be done.

Acknowledgments

We are especially indebted to Paul Clements who, as lead SCR software engineer, patiently assisted CRF validators in resolving problems that were encountered. Another contributor was Ms. Kathryn 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.
References


L. Chmura
Naval Research Lab
21 of 63


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

1. Introduction

There are two visual indicators controlled by the OFP on the A-7 aircraft: one that can, and one that cannot, be seen by the pilot during flight. These are currently labeled "MS Non-aligned" and "Auto-Cal", respectively. Each can be set steady, on blinking, or off.

2. Interface overview

2.1. ACCESS PROGRAM TABLE

<table>
<thead>
<tr>
<th>Program</th>
<th>Parameters</th>
<th>Description</th>
<th>Unused events</th>
</tr>
</thead>
<tbody>
<tr>
<td>4G/LAUTOCAL.INDICATOR+</td>
<td>p1: VIS.ind.lead;</td>
<td>Off 1 1 Auto-cal+</td>
<td>None</td>
</tr>
<tr>
<td>4S/LAUTOCAL.BLINK RATE+</td>
<td>p1: real; 1</td>
<td>blink/sec</td>
<td></td>
</tr>
<tr>
<td>4G/LNON_ALIGN.INDICATOR+</td>
<td>p1: VIS.ind.lead;</td>
<td>Off 1 1 Non-align+</td>
<td></td>
</tr>
<tr>
<td>4S/LNON_ALIGN.BLINK RATE+</td>
<td>p1: timeinc, 1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

4IL. AUTOCAL.INDICATOR+

ks: 8008 THEN "Auto-Cal" indicator turned on; IF p1=8018 THEN "Auto-Cal" indicator turned off; If p1=Hintermittent THEN "Auto-Cal" indicator turned on and off at the rate set by 4S/LAUTOCAL.BLINK RATE; or at the system default rate.

4S/LAUTOCAL.BLINK RATE+

When commanded to blink, the blink time will be p1.

4S/LNON_ALIGN.INDICATOR+

ks: 8008 THEN "Non-Align" indicator turned on; IF p1=8018 THEN "Non-Align" indicator turned off; If p1=Hintermittent THEN "Non-Align" indicator turned on and off at the rate set by 4S/LNON_ALIGN.BLINK RATE; or at the system default rate.

4S/LNON_ALIGN.BLINK RATE+

When commanded to blink, the blink time will be p1.

3. Local type definitions

4S/LAUTOCAL.INDICATOR+

Enumerated: 8008, 8018, Hintermittent

4S/LNON_ALIGN.INDICATOR+

The state of the non-align indicator as last set by 4S/LNON_ALIGN.INDICATOR+

4S/LAUTOCAL.INDICATOR+

The state of the auto-cal indicator as last set by 4S/LAUTOCAL.INDICATOR+

4S/LNON_ALIGN.INDICATOR+

The state of the non-align indicator as last set by 4S/LNON_ALIGN.INDICATOR+

5. Undesired event dictionary

None.

6. System generation parameters

#Auto-cal blink default#* Type: timeinc. Default blink interval for "Auto-Cal" indicator.

#Auto-cal init state#* Type: VIS.ind.lead. The system-load-time value for 4S/LAUTOCAL.INDICATOR+

#Non-align blink default#* Type: timeinc. Default blink interval for "Non-Align" indicator.

#Non-align init state#* Type: VIS.ind.lead. The system-load-time value for 4S/LNON_ALIGN.INDICATOR+

* The values of this system generation parameter may be set by user software. See section 1.1 of the introduction to this document.

Figure 1: Example of Module Interface Specification
Figure 2: Completed CRF Form
Figure 3: CRF Accumulation

Figure 4: Cumulative Effort in Resolving CRFs

L. Chmura
Naval Research Lab
29 of 63
Figure 5: Rejected CRFs: Percentage of Total

Total CRFs: 325

Date of Resolution

Figure 6: Rejected CRF Resolution Effort: Percentage of Total

Total: 400.6 hours

Date of Resolution

L. Chmura
Naval Research Lab
30 of 63
Figure 7: Error Corrections: Percentage of Accepted Changes

Total Changes: 297

Date of Resolution

Figure 8: Error Correction Effort: Percentage of Accepted CRF Resolution Effort

Total Change Effort: 382.8 hours
Figure 9: Correction or Completion Errors: Percentage of Error Corrections

Total Error Corrections: 199

Date of Resolution

Figure 10: Accepted CRFs Categorized by Resolution Effort

Total CRFs: 297

- Hour or Less  --- Hour < ... ≤ Day  --- Day < ... ≤ Week
Figure 11: Error Corrections Categorized by Resolution Effort

Figure 12: Cumulative Average CRF Effort
Figure 13: Duration of an Error in the System

Figure 14: Accepted CRFs Categorized by Number of Modules Changed
Figure 15: Accepted CRFs Categorized by Number of Interfaces Updated

Figure 16: Error Causes: Percentage of Total
Figure 17: Ratio of Cumulative CRF's Over Cumulative Origination Activity Hours

Figure 18: Ratio of Cumulative CRF Resolution Effort by Origination Activity Over Cumulative Activity Hours
Figure 19: Ratio of Cumulative Error Corrections and Accepted CRFs to Cumulative Project Months

Figure 20: Date of Origin PIR for AT
Figure 21: Date of Origin PIR for DI

Figure 22: Date of Origin PIR for EC
Figure 23: Date of Origin PIR for FD

TOTAL CRFs: 27

Date of Origin

January 1978 to January 1984

Figure 24: Date of Origin PIR for SS

TOTAL CRFs: 6

Date of Origin

January 1978 to January 1984

L. Chmura
Naval Research Lab.
39 of 63
Figure 25: Date of Resolution PIR for AT

TOTAL CRFs: 2

Figure 26: Date of Resolution PIR for DI

TOTAL CRFs: 11

L. Chmura
Naval Research Lab
40 of 63
Figure 27: Date of Resolution PIR for EC

TOTAL CRFS: 119

Date of Resolution

Figure 28: Date of Resolution PIR for FD

TOTAL CRFS: 27

Date of Resolution
Figure 29: Date of Resolution PIR for SS

TOTAL CRFs: 6

Date of Resolution

L. Chmura
Naval Research Lab
42 of 63
CHAPTER 23

DI.WOG: WEIGHT ON GEAR SENSOR

1. Introduction

The weight on gear device is a sensor that detects whether or not the aircraft is resting on its landing gear. This data can be used to infer whether or not the aircraft is airborne.

2. Interface overview

2.1. ACCESS PROGRAM TABLE

<table>
<thead>
<tr>
<th>Program</th>
<th>Parameters</th>
<th>Description</th>
<th>Undesired events</th>
</tr>
</thead>
<tbody>
<tr>
<td>+G_WEIGHT_ON_GEAR+</td>
<td>p1: boolean; O</td>
<td>!+WOG+!</td>
<td>None</td>
</tr>
</tbody>
</table>

2.2. EVENTS SIGNALED

@T/@F/=T/=F(!+WOG+!)

3. Local type definitions None.

4. Dictionary

!+WOG+! $\text{true}$ if weight on landing gear detected.

5. Undesired event dictionary None.

6. System generation parameters None.