TECHNICAL SUMMARY
1982

REPORT TO THE NATIONAL AERONAUTICS
AND SPACE ADMINISTRATION

NSG-5123

Department of Computer Science
University of Maryland
College Park, MD 20742

Principal Investigator:
Dr. Victor Basili

(NASA-CR-169476) SOFTWARE ENGINEEFING
LABOEATOBY (SEL) REFCRT TO THE NATICNAL
AERONAUTICS AND SPACE ADMINISTEATICN
(Maryland Univ.) 17 p HC A02/MF A01
CSCL 09B G3/61

N83-12880

Unclas
01049

Overview

During 1982, in conjunction with NASA/GSFC Software Engineering Laboratory (SEL), research was conducted in 4 areas: Software Development Predictors, Error Analysis, Reliability Models and Software Metric Analysis. Summaries of the projects follow below.

1. Software Development Predictors

A study is being done on the use of dynamic characteristics as predictors for software development. It is hoped that by examining a set of readily available characteristics, the project manager may be able to determine such things as when a project is in trouble and evaluate the quality of the product as it is being designed.

Project DEB was selected as the control for the project since it was considered fairly successful and is well documented. Information found in the history files and resource summary files was initially utilized. These files were chosen because the information they contain is readily accessible to the manager (ie. number of lines of code, manpower, computer time, etc.). Several profiles of project DEB were then made using this information. Project DEA's profiles were then compared with these results. This project was chosen because it was very similar to DEB but was considered less successful.

The history file was first examined to see if any growth pattern existed for the lines of code. The initial look at DEA and DEB looked hopeful but further investigation of other projects showed no discernible pattern. Other examinations of this file yielded similar results.

When a comparison of the information in the history and resource summary files was made some differences did appear. Initial plots used accumulative totals versus different time factors. These plots did demonstrate visible differences between the two projects. Further investigation using weekly totals instead of accumuiative totals showed an even larger difference between the projects.

Project DEA had a higher frequency of changes at the beginning of the project, while at the same time, the number of hours of manpower reported for the interval was less. The number of computer runs made was higher for DEB in the part of the project where DEA was experiencing the higher number of changes per manpower. In all, project DEA appears to have had less effort placed during the early phase of the project which may of led to the problems in the end. Another important aspect of project DEA was that several thousand lines of code appear to have been transported. Adaptation of this code may explain the high number of changes initially seen in DEA.

From this examination the following general goals and hypothesis have been generated:

A) The manpower usage in the SEL environment is a discernible pattern and may be used as a predictor.

1) The ideal staffing for a successful project is a two hump curve with the second hump beginning roughly 2/3 into the project.

2) The two humps mentioned in hypothesis 1 should peak at approximately the same height.

3) The maximum peak height of the first hump is proportional to the final size of the project. This also hold for the second hump based on hypothesis two.

4) The location of the two peaks is constant with relation to the amount of manpower utilized.

5) The amount of manpower expended between the two peaks is constant.

6) Projects deemed less successful by subjective analysis have sharp changes in the amount of manpower spent per change.

B) The pattern of changes in relation to manpower, computer runs, lines of code, etc. may be used as a predictor in the SEL environment.

1) The amount of manpower to make a change should increase toward the end of a project and be stable at the beginning.

2) The manpower per change should be lower in the beginning of the project. See also goal D.

3) Projects deemed less successful by subjective analysis have sharp changes in the amount of manpower spent per change.

4) The ratio of changes to computer run should decrease as the project evolves.

5) The amount of computer time spent on detecting and correcting a given change will remain constant.

C) The number of computer runs is closely related to the development of a project and may be used to judge project development.

1) The number of computer runs remains constant during the initial hump of the staffing curve. The number of computer runs will drop during the second hump of the staffing curve.

2) The ratio of changes to computer runs should decrease as the project evolves.

D) A close examination of the types of changes and the pattern they make over time should be a good indication of the success of a given project.

1) Time consuming changes that occur late in the project more often appear in modified code.

2) Unit testing is not as extensive on modules with modified code. Undetected errors may cause major problems latter in development.

3) The types of changes vary across the development of a project.

4) The number of changes per hour of manpower is related to the type of changes being done.

5) The types of change that require more time to correct occur during the second staffing hump.

Several projects will now examined to test the validity of these finds. The change report forms will also be examined to see if the information in them yields any useful predictors.

To conclude, the study has completed its initial analysis of the two projects. It appears there are some significant factors that could be useful as predictors. Further analysis may yield some information that would be useful to a project manager.

2. Error Analysis

A). Publication of existing results -- Three papers are being prepared from earlier work on error analysis conducted by the SEL laboratory. One is on the data collection methodology and the validation of the accuracy of the data, the second one is on the analysis of the SEL projects directly and the third one is a comparison of the SEL projects with projects of the Naval Research Laboratory. These papers are currently being submitted for publication and will be published as University of Maryland Technical Reports in the interim.

B). A study on software errors and complexity -- The distribution and relationships derived from the change data collected during the development of the medium scale satellite project shows that meaningful results can be obtained which allow insight into software traits and the environment in which it is developed The project studied in this case was GMAS. Modified and new modules were shown to behave similarly. An abstract classification scheme for errors which allows a better understanding of the overall traits of a software project was also provided. Finally, various size and complexity metrics are examined with respect to errors detected within the software yielding some interesting results. A University of Maryland Technical Report describing these results was published [Bas82]. This paper has been submitted for publication.

C). A further examination of the error characteristics of the DE_A and DE_B projects is currently being undertaken. This error analysis is being conducted using the techniques developed and documented in [Wei81] and [Per82]. The focal point of this research effort is to characterize errors in the NASA/GSFC software development environment.

A preliminary review of a sample of the Change Report Forms from both DE_A and DE_B has been conducted. The sample included only those CRF's for which an error change was reported. The purpose of this review was to 'get a flavor' for the data collected and to preliminarily assess the consistency of that data with the results found to date by SEL personnel.

The sample included 98 CRF's from DE_A and 90 CRF's from DE_B. Of the 98 CRF's from DE_A, 63 (64.3%) of the errors were classified as an 'error in the design or implementation of a single component.' Of the 90 CRF's from DE_B, 16 errors were reported as 'clerical errors.' Of the remaining 74 DE_B errors (non-clerical errors), 61 (84.2%) of the errors were also classified as 'errors in the design or implementation of a single component.'

Although the percentage classified as 'errors in a single component' for DE_B was higher than the other studies, these preliminary results appear to follow the results of previous analyses [Wei81]. As in that previous work, the distribution of errors in other categories does not neatly fit a pattern. In fact, there are too few events in the other categories to draw any initial conclusions. It will be interesting to explore the reason(s) DE_B experienced a substantially larger number of 'clerical errors.'

There are marked differences in the remaining DE_A and DE_B error reports. This may be attributable to the reported differences in the two projects. It is not possible at this time to conjecture on more tangible causes for the differences. The full set of error change reports will have to be examined, for both projects.

It is worth noting here that for DE_A, 31 of 98 error reports (31.6%) examined were classified as being an 'error in the design or implementation of more than one component.' Based on previous results cited above, this is an unusually high percentage. Only 4 components (4.1%) had errors reported that were not in the design or implementation of component(s) categories.

As part of the preliminary work toward the above goal, the related literature released by SEL was reviewed. A conclusion reached was that the definitions of several critical terms were not necessarily consistent, and often times the technical reports make too great an assumption about the uniformity of use of software engineering terms.

'Interface' provides a good example of an ill-defined yet oft used term. Using the definition from [Wei81] (the same definition is used in [Bas80b] and [Glo79]) it is arguable that interface errors can be captured five ways from the CRF:

-an error involving more than one component;

-an error involving a common routine;

-from textual comments in the CRF (eg: a CRF for which the error was entered as having affected one component but the text indicated that the error was in a subroutine call statement);

-an error reported as having been located in one component but the change required to repair the error affected more than one component; and

-a change that caused an error because either the change invalidated an assumption made elsewhere in the software or an assumption made about the rest of the software in the design of the change was incorrect (contingent on ability to capture supporting text and ability to distinguish from erroneous assumptions made about a single component).

An effort is currently underway to develop a more restrictive set of definitions for software engineering terms, specifically those that apply to error analysis. The basis of this effort is the set of definitions published in [Bas80] and [Glo79] and will be modified, as necessary, in consultation with those persons associated with SEL in the past and present, whose work is or was related to the error analysis effort.

3. Reliability Models

A study is being performed in the area of reliability models. This research includes the field of program testing because the validity of some reliability models depends on the answers to some unanswered questions about testing.

The eventual goal of this research is to understand how and when to use reliability models. We are investigating the use of functional testing because some reliability models make assumptions about the way program testing is accomplished [Musa]. It is not known if functional testing satisfies the random testing assumptions made by the reliability models. The validity of reliability models that use data generated by functional testing is uncertain until this question is answered.

We are using structural coverage metrics to gain further insight into the effects of functional testing. A structural coverage metric is a measure of how much of a program was executed for given input data. Studying the coverage metric may allow us to develop other measures of reliability.

An additional bonus of this research is that it allows us to compare functional testing and structural testing. It is not known how these two methods of testing are related. The results of this investigation may answer that question.

Since January background material has been studied with regard to reliability models, and functional and structural testing [Mueller]. A FORTRAN preprocessor has been written to calculate the structural coverage metrics of GSFC FORTRAN source code.

The preprocessor calculates the simplest metric, the percent of executable code that is executed. There are several ways to measure coverage [Auerbach]. One method uses interpretation of the source code. The interpreter records which statements are executed. At the end of interpretation, it writes a list of executed statements.

The second method uses "switches", small sections of code that are inserted into the source program text wherever the flow of control diverges or converges. The switch has 2 values: 0 if it was not executed, 1 if it was executed. The value of the switches is output after execution.

An example:

```
INTEGER SWITCH ( N )

FOR I = 1, N
SWITCH (I) = 0
          .
          .
          .
READ ( J );
IF ( even ( J ))
        THEN
                SWITCH ( 1 ) = 1;
                .
                .
                .
        ELSE
                SWITCH ( 2 ) = 1;
                .
```

```
                    .
                    .
          ENDIF
          .
          .
          .
    FOR I = 1, N
          WRITE ( SWITCH ( I ));
    END
```

When this program is executed, one of the two branches of the if statement will be executed. By examining the values of the array SWITCH, we can determine what code was executed. By analyzing the code and counting statements, the number of statements executed can be determined. In practice, the amount of data generated will be large. Software tools are needed to help analyze the data.

The switches can be inserted by a preprocessor (before compilation) or by a compiler (during compilation). The switches may be in-line code (as in the example) or a call to a switch subroutine that records the flow of control.

This latter approach was taken and a preprocessor was developed that runs on VAX/Unix at UMCP. The preprocessor takes a copy of the input source code, and modifies it. This modified copy will be returned to the source computer (at GSFC) where it will be compiled and executed. The execution produces the desired coverage data. The coverage data will be returned to the University for analysis.

Many things remain to be done before we reach our goal of understanding how and when to use reliability models.

The immediate goal is to try to answer the functional testing / reliability model question. The project RADMAS has been chosen as an experimental system [CSC]. The preprocessor must be used to modify the RADMAS source code. (The RADMAS project and its functionally-generated acceptance tests have been made available for the coverage experiment.) The modified RADMAS code must be executed at GSFC using the functionally-generated acceptance tests.

This experiment should answer these questions about functional testing and reliability models:

-What is the percent coverage of functional testing?

-Does functional testing meet the randomness requirements

of the MTTF models? If not, can it be made to?

-Do the structural metrics show any useful patterns in the way that functional testing tests programs? How does the coverage set grow? At what rate does the coverage set grow?

-How independent are individual tests from a coverage point of view?

The results of this experiment will raise further questions about functional testing and reliability models. This will require more experimentation. If these questions are answered, there is more work to do concerning how and when to use reliability models.

4. Software Metrics.

The attraction of the ability to predict the effort in developing or explain the quality of software has led to the proposal of several theories and metrics [Hal77, McC76, Gaf, Che78, Cur79]. In the Software Engineering Laboratory, the Halstead metrics, McCabe's cyclomatic complexity and various standard metrics have been analyzed for their relation to effort, development errors and one another [Bas82a]. This study examined data collected from seven SEL (FORTRAN) projects and applied three effort reporting accuracy checks to demonstrate the need to validate a database.

The investigation examined the correlations of the various metrics with effort (functional specifications through acceptance testing) and development errors (both discrete and weighted according to amount of time to locate and fix) across several projects at once, within individual projects and for individual programmers across projects.

In order to remove the dependency of the distribution of the correlation coefficients on the actual measures of effort and errors, the non-parametric Spearman rank-order correlation coefficients were examined [Ken79]. The metrics' correlations with actual effort seem to be strongest when modules developed entirely by individual programmers or taken from certain validated projects are considered. When examining modules developed totally by individual programmers, two averages formed from the proposed validity ratios induce a statistically significant ordering of the magnitude of several of the metrics' correlations. The systematic application of one of the data reliability checks (the frequency of effort reporting) substantially improves either all or several of the projects' effort correlation with the metrics. In addition to these relationships, the Halstead metrics seem to possess reasonable correspondence with their estimators, although some of them have size dependent properties. In comparing the strongest correlations, neither Halstead's E metric, McCabes' cyclomatic complexity nor source lines of code relates convincingly better with effort than the others.

The metrics examined in this study were calculated from primitive measures derived from a source analyzing program (SAP -- Revision I) [Dec82]. An earlier version of this static analyzer implemented a less comprehensive definition of Halstead operators and operands[O'Ne78]. Some work has been done comparing the metrics' correlations when they have been determined from the different interpretations of the primitive measures.

This investigation has been submitted for publication to the Transactions on Software Engineering and will appear as a University of Maryland Technical Report.

5. References

[Auerbach] Auerbach Publishers Inc., "Practical Measures for Program Testing Thoroughness", 1977.

[Bas80] V. Basili, Tutorial on Models and Metrics for Software Management and Engineering, p. 340, IEEE 1980

[Bas82a] V. Basili, R. Selby and T. Phillips, "Data Validation in a Software Metric Analysis of FORTRAN Modules," -- to appear IEEE Transactions on Software Engineering, July 1982.

[Bas82b] V. Basili and B. Perricone, "Software Errors and Complexity: An Empirical Investigation," The Software Engineering Laboratory, University of Maryland Technical Report TR-1195, August 1982

[Bas82c] V. Basili, "An Assessment of Software Measures in the Software Engineering Laboratory," presented at Goddard Space Flight Center, January 1982.

[Card82] Card, D., F.McGarry and J. Page, "Evaluation of Management Measures of Software Development," Vol I & II, Software Engineering Laboratory Series, SEL - 82 - 001, Goddard Space Flight Center, September 1982.

[Chen 78] E. T. Chen, "Program Complexity and Programmer Productivity," IEEE Transactions on Software Engineering, Vol. SE-4, No. 3, pp. 187-194 (May 1978).

[CSC] Computer Sciences Corporation, RADMAS User's Guide., September 1981.

[Curtis et al 79] Curtis, Sheppard and Milliman, "Third Time Charm: Stronger Prediction of Programmer Performance by Software Complexity Metrics," Proceedings of the Fourth International on Software Engineering, pp. 356-360 (1979).

[Decker & Taylor 82] W. J Decker and W. A. Taylor, "FORTRAN Static Source Code Analyzer Program (SAP) User's Guide (Revision 1)," SEL-78-102, Software Engineering Laboratory, (May 1982).

[Gaffney & Heller] J. Gaffney and G. L. Heller, "Macro Variable Software Models for Application to Improved Software Development Management," Proceedings of Workshop on Quantitative Software Models for Reliability, Complexity and Cost, IEEE

Computer Society.

[Glo79] S. Gloss-Soler, The DACS Glossary -- A Bibliography of Software Engineering Terms, Data and Analysis Center for Software, p. 56, October 1979

[Halstead 77] M. Halstead, Elements of Software Science, Elsevier North- Holland, New York (1977).

[Kendall & Stuart 79] M. Kendall and A. Stuart, The Advanced Theory of Statistics, Vol. 2, Fourth Ed., MacMillian, New York, 1979, pp. 503-508.

[McCabe 76] T. J. McCabe, "A Complexity Measure," IEEE Transactions on Software Engineering, Vol. SE-2, pp. 308-320 (December 1976).

[Mueller] Mueller, Barbara, "Test Data Selection: A Comparison of Structural and Functional Testing", April 1980, private paper.

[Musa] Musa, John, D., "Software Reliability Management", Software Life Cycle Management Workshop, August 1977.

[O'Neill et al 78] E. M. O'Neill, S. R. Waligora and C. E. Goorevich, "FORTRAN Static Source Code Analyzer (SAP) User's Guide," SEL-78-002, Software Engineering Laboratory (February 1978).

[Pic82] G Picasso, "The Rayleigh Curve as a Model for Effort Distribution Over the Life of Medium Scale Software Systems," Department of Computer Science, University of Maryland Technical Report TR-1186, July 1982.

[Wei81] D. Weiss, "Evaluating Software Development by Analysis of Change Data," The Software Engineering Laboratory, University of Maryland Technical Report TR-1120, November 1981