1992

NASA/ASEE SUMMER FACULTY FELLOWSHIP PROGRAM

MARSHALL SPACE FLIGHT CENTER
THE UNIVERSITY OF ALABAMA

USING SOFTWARE METRICS AND SOFTWARE RELIABILITY MODELS TO ATTAIN ACCEPTABLE QUALITY SOFTWARE FOR FLIGHT AND GROUND SUPPORT SOFTWARE FOR AVIONIC SYSTEMS

Prepared By: Stella Lawrence

Academic Rank: Professor

Institution: Bronx Community College,
Department of Engineering Technologies

NASA/MSFC:

Office: Information & Electronic Systems Lab.
Division: Software & Data Management
Branch: Systems Software

MSFC Colleague(s): Kenneth Williamson
Charles Cozelos

This paper is concerned with methods of measuring and developing quality software. Reliable flight and ground support software is a highly important factor in the successful operation of the space shuttle program. Reliability is probably the most important of the characteristics inherent in the concept of "software quality." It is the probability of failure free operation of a computer program for a specified time and environment.

There has been an increased awareness in recent years of the critical problems that have been encountered in the development of large scale software systems. These problems not only include the cost and schedule overruns typical of development efforts, and the poor performance of the systems once they are delivered, but also include the high cost to maintain the systems, the lack of portability and the high degree the systems can be sensitive requirement changes.

The efforts related to the development of a standard programming language, and of software development tools and aids, all provide partial solution to the above problems by encouraging disciplined development of software and therefore a controlled development process.

Recently there has been a great deal of research in the area of software metrics. A number of metrics which measure various attributes of software and relate them to different aspects of software quality have been developed and evaluated. The program manager responsible for the development of the software can establish specific software product quality goals and measure the progress towards these goals during development. Metrics can also provide the means to assess the difficulty in modifying a software product.

A validated metric is one whose values have been statistically associated with corresponding quality factor values or attributes of software that contribute to the quality of the software. The critical value of a validated metric is that value which is used to identify software which has unacceptable quality. The major benefit of validation is that it increases the probability that the metric will be a good indicator of quality. The six criteria that a metric must satisfy to be validated are associativity, consistency, discriminative power, tracking, predictability and repeatability.

The quality factor reliability has been associated with various subfactors. These include completeness, accuracy, consistency, error tolerance, simplicity, availability, nondeficiency and anomaly management.

The measurement of software complexity is receiving increased attention since software accounts for a growing proportion of total computer system costs. Besides the

above mentioned complexity measures, the following basic and conceptually simple measures are also considered: the total number of lines encountered in the main body of the program, the number of lines of code, the total number of characters, the number of code characters, then number of comments and the number of comment characters in a program.

Parsers have been developed and used to compute the various metrics for Pascal and Fortran programs, and recently for Ada programs. The Dynamics Research Corporation has developed a Measurement and Analysis Tool called AdaMat. AdaMat is a specific source code quality analysis tool. Its metrics hierarchy is based on the RADC (Rome Air Development Center) metrics framework. It measures adherence to over 240 quality principles which impact the reliability, maintainability and portability of Ada source code. Its principles are based on the most effective use of Ada language features and adherence to long standing software engineering principles. In the AdaMat system the quality factor, Reliability, has two subfactors, Anomaly Management and Simplicity. Anomaly Management has three subfactors: Prevention, Detection and Recovery. Simplicity has three subfactors: Coding Simplicity, Design Simplicity and Flow Simplicity. AdaMat reports contain concise information on each metric's level of adherence, the ranking option allows worst case Ada units to be isolated rapidly, and AdaMat Documentation clearly explains what is being counted by each metric. It also explains why it is important to quality, provides a suggested method of improvement and gives a source code example of adherence to the metric principle and non-adherence to the principle.

AdaMat provides a user friendly interface which makes reporting simplified and straightforward, it provides maintenance cost savings resulting from the use of a disciplined development process. In addition there are substantial early error detection cost benefits. Lastly it provides an automated code review and inspection process. In particular, formal source code inspections can use such a tool to locate potential problem areas for the inspectors.

The second part of this project dealt with the development and calibration of quantitative models for predicting the quality of software. A software reliability model specifies the general form of the dependence of the failure process on the principal factors that affect it: fault introduction, fault removal, and the environment. Software reliability models are generally formulated in terms of random processes. Analytic expressions can be derived for the average number of failures experienced at any point in time, the average number of failures in a time interval, the failure intensity at any point in time, and the probability distribution of failure interval models. A good software reliability model gives good predictions of future failure

behavior, estimates MTTF, estimates time to test completion, is simple, widely applicable, and based on sound assumptions. Prediction of future failure behavior assumes that the values of the model parameters will not change for the period of prediction.

The models used in the present study consisted of:

1.SMERFS (Statistical Modeling and Estimation of Reliability Functions for Software). There are ten models in SMERFS: error count models (generalized Poisson model, NHPP model, Brooks and Motley, Schneidewind model, S-shaped reliability growth model) and time-between-error models (Littlewood and Verrall Bayesian model, Musa execution time model, geometric model, NHPP model for time between error occurrence, Musa logarithmic Poisson execution time model).

2. Kenneth Williamson's NHPP Binomial type software reliability model

3. Goel-Okumoto NHPP model.

The software utilized consisted of the IMCE (Image Motion Compensation Electronics) software flight data, BATSE (Burst Transient Source Experiment, Gamma Ray Observatory) software, and a further program will also utilize POCC (Payload Operations Control Center for the Space Shuttle), Payload Checkout Unit Software for the Space Shuttle and HIT's Software (Space Shuttle Telemetry Systems).

Before discussing the results obtained with the models used in the present study, it must be kept in mind that software reliability modeling is just one of many tools. It cannot provide all the answers to the problems managers and developers must face. It must be taken as a bit of information, which along with others, is helpful in making a realistic judgment concerning a program's status.

For a first run, the results obtained in modeling the cumulative number of failures versus execution time showed fairly good results for the data. Plots of cumulative software failures versus calendar weeks were made and the model results were compared with the historical data on the same graph. If the model agrees with actual historical behavior for a set of data then there is confidence in future predictions for this data.

Considering the quality of the data, the models have given some significant results, even at this early stage. With better care in data collection, data analysis, recording of the fixing of failures and CPU execution times, the models should prove valuable in making predictions regarding the future pattern of failures, including an estimate of the number of errors remaining in the software and the addition-

al testing time required for the software failure rate to reach a chosen target level. In addition, conditions occurring during V&V are not always covered by the models. A Center Director's Discretionary Fund proposal is planned to address these conditions.

It appears that there is no one "best" model for all cases. It is for this reason that the aim of this project was to test several models. One of the recommendations resulting from this study is that great care must be taken in the collection of data. When using a model, the data should satisfy the model assumptions.

As previously stated, the data has to have the ability to correctly identify and measure what is desired. The data provided must satisfy the following:

1. It should be correctly recorded.

2. It should consist of samples that are random in nature.

3. It should be stated in CPU hours between failures or error counts per interval.

4. All error failure data should be accurate.

Reliability will improve if the field software is corrected as failures occur. What about repeated failures due to the same fault? Fixing of faults leading to failures has to be properly recorded and properly attended to. The record of failures must be obtained for a sufficient length of time. Recent theory indicates that the failure intensity function probably decreases exponentially with time, i.e. a plot of the rate of occurrence of failures versus the number of faults found decreased asymptotically to zero.

There are also several recommendations regarding the use of the models:

1. The models require the insertion of various parameters. The models should be run with various values of these parameters, which should be carefully chosen for optimum results.

2. The data should be modeled piecewise, in addition to running the models for the total data.

3. Various forms of data input are provided including time between failure data and error count data and the model may yield different results for different types of data input.

4. The length of the trial should be a proportion of the expected life of the system; predictions made from a very small set of data tend to be poor.

5. The rate of manifestation of errors varies greatly from fault to fault, models which treat all faults as having the same rate may lead to optimistic bias estimates. Perhaps some type of analysis should be performed to classify failures by severity, what kind of failure is it and is it critical or not?

To sum up the preliminary trials indicate that the models tested show much promise and that with their proper use and tailoring they are expected to yield an accurate reliability prediction for the flight and supporting ground software of embedded avionic systems.

REFERENCES

1. Musa, John S., Iannino, A., and Okumoto, Kazuhira, Software Reliability Measurement, Prediction, Application, McGraw-Hill Book Company, New York, 1987.

2. Schneidewind, Norman F., Validating Software Metrics Naval Surface Warfare Center, Dahlgren, VA 22448, Sept. 1990.

3. Dynamics Research Corporation, AdaMat, Measurement and Analysis Tool, 60 Frontage Road, Andover, MA 01810, Oct. 1991

4. Farr, William H., Strategic Systems Department, Naval Systems Warfare Center, Dahlgren, VA 22448, Sponsored by Strategic Systems Programs, Washington, DC 203765-5002, Statistical Modeling and Estimation of Reliability Functions for Software (SMERFS), Report No. NSWC TR 84-373, Revision No. 2, March, 1991.

5. Williamson, Kenneth, Non-Homogeneous Poisson Process Binomial Type Software Reliability Model, Preliminary Edition, EB41/Software & Data Management Division, EB42/Systems Software Branch, Marshall Space Flight Center, Huntsville, AL 35812, July, 1991.