SPACE STATION SOFTWARE RELIABILITY ANALYSIS BASED ON FAILURES OBSERVED DURING TESTING AT THE MULTISYSTEM INTEGRATION FACILITY

Final Report

NASA/ASEE Summer Faculty Fellowship Program-1987

Johnson Space Center

Prepared by:	Tak Chai Tamayo
Academic Rank:	Assistant Professor
University Department:	University of Houston-UP Department of Industrial Engineering Houston, Texas 77004

NASA/JSC

Directorate:	Mission Support
Division:	Spacecraft Software Division
Branch:	Systems Development
JSC Colleague:	Richard E. Coblentz
Date:	August 14,1987
Contract Number:	NGT 44-001-800

ABSTRACT

Quality of software not only is vital to the success operation of the Space Station, it is also an important factor in establishing testing requirements, time needed for software verification and integration as well as launching schedules for the Space Station. Defense of management decisions can be greatly strengthened by combining engineering judgements with statistical analysis. Unlike hardware, software has the characteristics of no wearout and costly redunduncies, thus making traditional statistical analysis not suitable in evaluating reliability of software.

A statistical model was developed to provide a representation of the number as well as types of failures occur during software testing and verification. From this model, quantitative measure of software reliability based on failure history during testing are derived. Criteria to terminate testing based on reliability objectives and methods to estimate the expected number of fixings required are also presented here.

INTRODUCTION

The purpose of Multisystem Integration Facility (MSIF) is to provide a facility on which information systems for the Space Station, which are produced by different developers, may be integrated, tested, verified, certified for flight, and packaged for launch. The MSIF concept was motivated by the facts that Space Station softwares are being developed by multiple developers at different sites. The systems are highly distributed and will be built up in phases, over a number of launches. Several upgrades and changes will take place over the life of the Space Station. MSIF will be required to first perform testing using computer models of all the Space Station systems. As real systems are delivered at MSIF, testing will be performed using combinations of models and real systems. The final test will be one in which all systems are actual flight-ready versions. Since the correction of errors found during multisystem integration is the responsibility of the developer, control over delivered systems may be returned to the developer for the correction of errors, and then back to the MSIF to continue testing.

Software is an important element of the Space Station, and is vital to its successful operation. Failure of softwares can be life-threatening in some cases. In addition, the quality of software can greatly affect the amount of fixings required during the testing, integration or verification process, thus making it possible to cause delays in launching of the Space Station, which is scheduled to begin in January, 1994. Consequently there is an urgent need to search for a quantitative measure of the reliability of the software, and to develop methods of combining reliability of software and hardware elements of

the Space Station to establish the system reliability of the Station. The concept of software reliability differs from that of hardware reliability in that failure is not due to a "wearing out" process. Software failures are in fact errors which, owing to hte complexity of a computer program, do not become evident until the combination of conditions bring the error to light. Unlike the hardware bathtub curve, there is no wearout characteristics, but only a continuing burn-in. Once a software error is identified and properly fixed, it is in general, fixed for all time. However, the large number of possible paths and its inputs in a space station software makes complete testing of the software generally impossible.

Several approaches are currently available for testing of a software: path testing, functional testing and formal proofs of correctness. A complete functional test would verify that the correct output is produced for each input . It would consist of subjecting the program to all possible input streams. However, a ten-character string has 2^{80} possible input streams and corresponding outputs. So complete functional testing in this sense is clearly impractical. In path testing, one would design a sufficient number of test cases to assume that every path through the routine is exercised at least once. But most often, even the number of paths through a small routine can be astronomical to permit all paths to be tested. As for formal proofs of correctness, each program statement is examined and used in a step of an inductive proof that the routine will produce the correct output as stated by formal mathematics. The practical issue here is that such proofs are very expensive and have been applied only to numerical routines. Not only are all known approaches to absolute demonstrations of error-free

impractical, they are impossible as well.

Because exhaustively tested and error-free software are not made possible by current acceptance procedures, purchaser of a software product is provided with no quantitative information on which to base an acceptance decision and is thus forced to make these decisions based mostly on intuition and his own experience in similiar situation. Therefore our goal should be to provide sufficient testing to assure that the probability of failure due to hibernating errors is sufficiently low to be acceptable. It is expected the level of testing required will depend on the system/component, criticality and complexity, state of development and cost and usage of the system.

Software reliability is defined here as the probability that a given software operates for some time period without software error detectable by executing the codes on the machine for which it was designed, assuming that it is used within design limits. Such being the case, test cases should be designed to cover the operating scenarios of the information system designed. When softwares are delivered to MSIF, they have already been successfully tested on the flight-compatible hardware. MSIF testing will start using models of other systems, and progress to using delivered versions of the other systems. Current concepts of MSIF requires if errors are detected, the software be returned to its developer with descrepancy reports of the errors found. After proper fixing, the software is returned to MSIF for retesting. In general, while fixing the errors found, new errors are also introduced. A portion of the old errors persisted, and will reoccur during the retesting. This process is repeated until a decision is made about the quality of the software based on the testing results. In the past it normally means the

software operates successfully on all test cases it was subjected to. The goals of this paper are:

1. Develop a statistical model to describe the failures behaviour during testing.
2. Obtain a statistical measure of software reliability, based on failure history observed during testing.
3. According to prespecified software reliability and failure history, establish criteria as to when testing of software can stop.
4. Combining reliability of software and hardware elements of the Space Station to establish "system" reliability of the Station.
5. Specified types of error records to be maintained during testing so that they can be used for later statistical analysis.

These goals are motivated by the fact that in the Space Shuttle program, the extent and degree of testings performed on space softwares have generally been made based on management judements. Verification requirements are to be determined individually for each hardware/ software product, based on criticality and risk associated with the hardware/software when it is integrated into the operational enviroment.

DESCRIPTION OF THE MODEL

The model of interested here is the failure behavior of a software after it is delivered to MSIF for testing and integration. A number of test cases designed to cover a selection of the enviroment in which the software will be used are run and errors occured during execution are recorded in descrepancy reports. The software is then returned to its developer for fixing. After proper fixing, it is returned to MSIF for retesting, where a portion of the old errors may reoccur and some new errors are detected.

Assumptions of the Model

1. All errors are caused only by the faults in the software, thus all others involved in testing are assumed to possess high fidelity.
2. All errors occure during testing are observed.
3. The number of new errors found are statistically independent of the total number of errors found during the previous trial.
4. The failure rate of new errors for each trial is dependent of the number of fixings already performed on the software.
5. The number of test cases run during each trial remains relatively constant.
6. All persisted errors are statistically independent of each other.
7. The number of new errors observed during each trial follows a poisson process.

Although it is possible that failure rate of new errors found

during each trial can be directly proportional, constant, or inversely proportional to the number of fixings performed, historical information gathered during the development of the Shuttle Orbiter primary flight software indicates a decreasing trend.

During each trial, the total number of errors detected consists of two independent entities: new and persisted errors.

Let

N_k = total number of errors detected during kth trial

X_k = number of new errors detected during kth trial, after (k-1) fixings by the developer

R_k = number of persisting errors from the previous trial.

Thus, the total number of errors detected at each trial is the sum of the number of new errors introduced by the last fixing and the number of errors persisted from the last trial, i.e.

$$N_k = X_k + R_k.$$

Analysis of the Model

Let

p_k = probability of an error found in kth trial to persist in (k+1)th trial

If the number of errors found during kth trial is n_k, then the probabaility of r errors persist in (k+1)th trial after fixing is:

$$\text{Prob}(R_{k+1} = r \mid N_k = n_k) = C(n_k,r)p_k^{\,r}(1-p_k)^{n_k-r}, \qquad r = 0,1,2, \ldots, n_k$$

where

$C(n_k,r)$ = the number of unordered samples of size r taken from n_k.

This is the conditional probability density of persisting errors based on the total number of errors found in previous trial and it follows a binomial distribution $B(n_k, p_k)$. As defined in the model, the total number of errors in the next trial is determined by the sum of two independent random variables, namely the numbers of new and persisted errors. This implies future errors is dependent of the number of errors found at present through persisted errors. Therefore the conditional probability density function of future errors based on present condition is:

$$\text{Prob}(N_{k+1} = n \mid N_k = n_k) = B(n_k, p_k)^* g(X_{k+1}).$$

where

$g(X_{k+1})$ is the probability density function of the number of new errors found during (k+1)th trial, and B*g represents the convolution of the two random variables, R_{k+1} and X_{k+1}.

By the assumption that X_{k+1} follows a poisson distribution with mean λ_{k+1}, the convolution of a binomial and poisson distributions is given as follows:

$$\text{Prob}(N_{k+1} = n \mid N_k = n) = \sum_{j=0}^{\min(n,n_k)} C(n_k, j) p_k^{\,j} (1-p_k)^{n_k - j} \exp(-\lambda_{k+1})(\lambda_{k+1})^{n-j}/j!$$

If p_k is relatively small, then the density function of $B(n_k, p_k)$ can be approximated with a poisson distribution, and the convolution of R_{k+1} and X_{k+1} is a poisson process given as:

$$\text{Prob}(N_{k+1} = n \mid N_k = n_k) \approx \exp(-\lambda_{k+1} - \mu_{k+1})\,(\lambda_{k+1} + \mu_{k+1})^n/n!$$

$$\text{with mean} = \lambda_{k+1} + \mu_{k+1} \quad \text{and} \quad \mu_{k+1} = n_k p_k.$$

In the case where n = 0, this conditional density function estimates the software reliability based on number of failures observed.

Criteria for Termination of Testing:

Extent of testing for softwares shall be conducted at a level consistant with its criticality level associated with the Space Station. Softwares that are highly critical to the successful operation of the Station will require high reliability, thus more detail testing than the others. Current concept documents of MSIF states thaat the degree to which a system is tested at MSIF depends on its risk category and how tightly coupled it is with the Data Management Systems (DMS). Suppose a particular software is required to have a minimum reliability level,

say REL, during the mission period T. In particular, REL is defined as the probability no error will occur during the mission period, and (1-REL) is the probability any error is detected during T. Hence the following inequality must be satisfied in order to meet the required reliability level REL.

$$REL \geq Prob(N_{k+1} = 0 \mid N_k = n_k)$$

Using the approximation of a poisson process, it is thus suffucient to solve for n_k^* such that:

$$REL \geq \exp(-\lambda_{k+1} - n_k p_k)$$

or by taking logorithm on both sides,

$$n_k^* \geq [\, -\ln(REL) - \lambda_{k+1} \,] / p_k \qquad (1)$$

Equation (1) gives the set (k, n_k^*) which yields the required reliability level REL. Since $n_k^* \geq 0$, it implies that the earliest time testing may terminate can be obtained by setting $n_k^* = 0$ and then solve for a smallest k that satisfies:

$$\lambda_{k+1} \leq -\ln(REL).$$

Let m be the expected number of fixings required before a software passes the testing and

$$m = E[k]$$

$$= \sum_{k=0}^{\infty} k \, \text{Prob(testing is terminated after k fixings)}$$

$$= \sum_{k=0}^{\infty} k \, \text{Prob}(N_k = n_k^*, n_k^* \geq 0)$$

which can be obtained by applying properties of conditional probabilities as follows:

$$\text{Prob}(N_k = n_k^*, n_k^* \geq 0)$$

$$= \sum_{n_{k-1}=0}^{\infty} \text{Prob}(N_k = n_k^*, n_k^* \geq 0, N_{k-1} = n_{k-1})$$

$$= \sum_{n_{k-1}=0}^{\infty} \text{Prob}(N_k = n_k^*, n_k^* \geq 0 \mid N_{k-1} = n_{k-1}) \, \text{Prob}(N_{k-1} = n_{k-1})$$

and

$$\text{Prob}(N_j = n_j) = \sum_{n_{j-1}=0}^{\infty} \text{Prob}(N_j = n_j \mid N_{j-1} = n_{j-1}) \, \text{Prob}(N_{j-1} = n_{j-1}) \, , \quad .$$

$$\text{for } j = 1, 2, \ldots, n_{k-1}.$$

Example

Suppose from historical data, one will observe X_k new errors after k fixings and X_k follows a poisson process with mean λ_k = 0.5exp(-k) per KSLOC[1] per hour in execution time. Ten percent of the errors found during a trial are corrected by the developer after one fixing. One hundred test cases which takes a total of 100 hours to execute are designed to test the Atmosphere Control and Supply (ACS) subsystem software, which provides total and partial pressure control within the pressurized habitation module in the Space Station. Such system consists of 10 KSLOC and is classified as criticality level 1 which requires a minimum reliability level of 0.999 during its useful life cycle of 20 years, i.e. the probability of no error being detected during the next 20 years of operation is 0.999.

By adjusting the unit of measurement, the mean number of failures for the ACS software during each trial period which lasts 100 hours is:

$$\lambda_k = 50\exp(-k)$$

The number of hours in 20 years = 175,200 and is equivalent to 1,752 trial periods.

To achieve reliability of 0.999 during the next 20 years, the software should achieve a minimum reliability level REL^* during the test period of 100 hours where

$$(REL^*)^{1752} = 0.999$$

or $$REL = 0.9999994$$

[1] KSLOC = thousand source line of codes

The minimum number of fixings needed for the ACS software based on the required relaibility level is given in Table 1. It shows that for required reliability of 0.9999994, the earliest time testing can terminate is after the 18th fixing when no error is detected during that trial. An estimate of the software reliability during the next period based on selected number of errors observed during testing are given in Table 2.

Table 1. The Minimum Number of Fixings Needed for the ACS Software Based on Required Reliability Level

Reliability Level (REL)	Minimum Number of Fixings (k)
0.90	6
0.95	6
0.99	8
0.999	10
0.9999	13
0.99999	15
0.999999	17
0.9999997	18
0.9999999	19

Table 2. Software Reliability Based on Number of Errors Observed

Trial (k)	# Errors Observed During Testing (N_k)	Reliability
1	0	0.0011514
1	1	0.0010418
2	0	0.0829635
2	1	0.0750685
3	0	0.4002035
3	1	0.3621191
4	0	0.7139821
4	1	0.6460377
5	0	0.8834349
5	1	0.7993650
6	0	0.9554297
6	1	0.8645085
7	0	0.9833667
7	1	0.8897870
8	0	0.9938485
8	1	0.8992713
9	0	0.9977325
9	1	0.9027858
9	2	0.8168743
10	0	0.9991652
10	1	0.9040821
10	2	0.8180473
11	0	0.9996928
11	1	0.9045595
11	2	0.8184792
12	0	0.9998869
12	1	0.9047351
12	2	0.8186382
13	0	0.9999584
13	1	0.9047998
13	2	0.8186967
14	0	0.9999847
15	0	0.9999943
16	0	0.9999979
16	1	0.9048355
17	0	0.9999992
18	0	0.9999997
19	0	0.9999999

C-3

Collection of Data

Certain data concerning the software will have to be collected in order to verify the statistical model described here. This includes:

1. The frequency of persisted errors from previous trial and new errors that are introduced during the fixing effort.
2. The number of fixings performed by the developer when these errors were found.
3. Criticality level of errors found at each trial.
4. A function which relates the failure rates of new errors at each trial with the number of fixings performed.
5. Probability distributions of the number of new errors detected at each trial.
6. Probability an identified error is corrected by the developer through one fixing.
7. Number of test cases applied on each trial.
8. Size of the software.

The function which relates failure rates of new errors with the number of fixings can be obtained by applying regression analysis on the frequency of failures obtained through historical data. Since a perfect fit of n experimental data may require a polynomial of degree (n-1), techniques of selecting a "sufficient" function may be needed to reduce this polynomial to an acceptable form. If distributions of failures are unknown, a Chi-square goodness-of-fit test can be employed to test for pattern of distributions. Data from the Space Shuttle softwares were not applied to this model because they made no distinction between persisted and new errors. However, the data did support a decreasing

trend with the number of fixings. When available, the amount of time and cost associated with each fixing can be combined with the amount of computer execution times required during testing to establish testing schedules and total cost of softwar testing so that launch schedules and budgets are met. Factor like degree of interaction with other systems is important in determining the reliability of a distributed system, and therefore should also be included in the data when available.

System Reliability

The Challenger accident shows that in case of accidents, defense of management decisions can be greatly strengthened if they are made based on combination of statistical analysis and engineering judgements. In the context of manned Space Station, it is important to explore reliability theory to assess the risk of extended human presence in space. Most Space Station systems are complex systems composed of hardware and software, both of which are required to be in operational states in order for the system to perform its designed function. It is thus necessary to include software as part of the components which form the reliability network of the Space Station. It has been shown that a system with subsystems and components in series will have reliability less than that of its weakest link. Suppose the ACS subsystem hardware in the habitation module has failure times which follows an exponential distribution with MTBF = 100 years and reliability of its software which interacts with the Data Management Systems (DMS) is 0.999 during the next 20 years. Then the ACS

subsystem will have reliability less than those of the hardware, software and DMS alone. In particular, if no scheduled maintenance work are to be performed during the next 20 years and the DMS has reliability of 0.995, then the reliability of the ACS subsystem is:

$$\begin{aligned} \text{Reliability} &= \exp(-20/100) \times 0.999 \times 0.995 \\ &= 0.8138224. \end{aligned}$$

This demonstrates an important fact that the reliability of a complex system decreases rapidly as more subsystems are added to the system design. For instance, a system which requires five components in series configuration will have reliability of 0.77 if each of the component was tested to have reliability of 0.95. Thus in order to achieve the goal of a highly reliable system, efforts should be made to obtain highly reliable hardware as well as software through either engineering design or testing. While traditional methods of redunduncy works well with hardware, it can be quite costly for complex software, as redunduncy in software normally means an independent development of the computer program. It is this unique characteristic of software which makes software testing an effective method to maintain software quality.

CONCLUSION

The lack of quantitative method to evaluate reliability of software delivered by the developer motivated the statistical model designed here. The unique characteristics of no wearout and costly redundancy has made software testing an only way besides software design to maintain software quality. The model developed here represents the failure pattern during software testing, which includes new errors introduced by the fixing and persisted errors from previous trial. Quantitative approaches were derived to predict the software reliability and criteria to terminate testing based on failure history. These results can be applied to enhance the safety of the Space Station and to avoid delays in launch schedules due to delay in the software verification process.

REFERENCES

1. Preliminary Space Station Multisystem Integration Facility Document, NASA JSC, June, 1987.

2. Musa, J. D., "A theory of Software Reliability and its Applications," IEEE Transaction on Software Engineering, Vol. SE-1, No. 3, pp. 312-327, September, 1975.

3. Musa, J. D., "Software Reliability Measurement," The Journal of Systems and Software, Vol. 1, pp. 223-241, Elsevier North Holland, Inc., 1980.

4. Final Report for "The Analysis of the Statistical and Historical Information Gathered During the Development of the Shuttle Orbiter Primary Flight Software," N82-29960, Sumitted by Texas A&M Research Foundation.

5. Feller, William, An Introduction to Probability Theory and its Applications, John Wiley & Sons, Inc.

6. Smith, David J., Reliability and Maintainability in Perspective, Practical, Contractual, Commercial and Software Aspects, Second Edition, John Wiley & sons, Inc.

7. Architectual Control Document for Enviromental Control and Life Support System, NASA JSC 30262, January, 1987.