8th National Computer Assurrance Conference (COMPASS), Gaithersburg, MD, June, 1993

RARE CONDITIONS - AN IMPORTANT CAUSE OF FAILURES

Herbert Hecht
SoHaR Incorporated
Beverly Hills, California

ABSTRACT

Data gathered by the author as well as those published by others show that rarely executed code has a much higher failure rate (expressed in execution time) than frequently executed code during the early operational period. More detailed analysis of the data shows that the inability to handle multiple rare conditions, such as response to hardware failures or exception conditions caused by the computer state, is a prominent cause of program failure in well-tested systems. Approaches to coping with this difficulty are discussed.

Background

About 10 years ago a number of investigators called attention to the increase in hardware and software failure rates during periods of high workloads [CAST81, IYER82, ROSS82]. In subsequent research the major cause of the increase in software failures was identified as faulty response to hardware exception or fault conditions [VELA83, 84].

The DSN Experiment

Motivated by these findings we investigated differences in failure rate between rarely executed code (RC) and frequently executed code (FC) in the NASA/JPL Deep Space Network (DSN). The overall purpose of this large software project is to retrieve spacecraft telemetry data and to uplink commands. Project personnel identified 5 segments each of RC and FC, the size and test results of which are shown in Table 1.

Table 1. Characteristics of DSN Code

Characteristic	FC	RC
Program size, KSLOC	185.1	144.3
No. of faults found in test	893	135
Fault density from test	0.0048	0.0016
Failures during first yr. operation	33	42
Failures during last 4 months of yr.	9	32

In addition to redundancy management and exception handling, the RC in this project included initialization and calibration routines. It was estimated that over a period of several months the FC would be exercised about 100 times as much as the RC. Thus, the failure rate (based on execution time) of the RC is over 100 times greater than that of the FC. Even if the estimate of the execution frequency is off by an order of magnitude, the difference in failure rate remains very significant. That the fault density from test for the RC was only one-third that of the FC indicates that the latter underwent much more intensive test. The comparison of failures during the last 4 months with those for the entire year shows that the rarely executed code also was debugged in operation at a much slower rate (which is understandable in view of the lower rate of usage).

A similar experience is reported in an article dealing with the development of a digital phone switching network in France [KANO87]. The software included routine switching code (called *Telephony* in the paper) which was executed almost constantly, and code supporting hardware redundancy management (called *Defense* in the paper) which was very rarely executed. The following experience is reported on the relative size of the code and the number of faults found. Some of the faults were found during code inspections and others were found by executing the code. The data do not permit a separate analysis of the two classes.

Table 2. Frequently and Rarely Executed Code in Phone Switch

CHARACTERISTIC	Frequently Executed	Rarely Executed
Size - fraction of total code	0.33	0.26
Fraction of total faults	0.29	0.30
Relative fault density - total	**0.88**	**1.15**
Global/total faults	0.11	0.20
Fraction of global faults	0.03	0.06
Relative fault density - global	**0.09**	**0.23**

In this case the overall fault density of the rarely used code was only slightly higher than that of the frequently used one, but among faults that caused globab l failures of the switching system there was a startling difference, with the rarely executed code accounting for the higher fault density among the most serious failures. There is no indication of the actual number of executions that each section was exposed to, but it can be assumed that the "defense" code saw only a minor fraction of the execution that was experienced by the frequently executed segments.

Space Shuttle Avionics Software

The failure data from the DSN and the French phone switching system permitted only a gross statistical analysis of possible failure causes. More recently failure data from the final (Level 8) testing of Space Shuttle Avionics software (SSA) became available to us that contained sufficient descriptive material on each failure to permit a much more detailed analysis. In this data set the distinction is drawn not between rare and frequently executed code but rather between rare events (RE) and normal events (NE) responsible for the failure. When at least one RE was responsible for the failure the corresponding failure report was classified as a rare event report (RR). The data were collected during the acceptance test interval for release 8B of the program, the first flight program immediately after the Challenger accident. The program had undergone intensive test prior to the period reported on here. NASA classifies the consequences of failure (severity) on a scale of 1 to 5, where 1 represents safety critical and 2 mission critical failures with higher numbers indicating successively less mission impact. During most of this period test failures in the first two categories were analyzed and corrected even when the events leading to the failure were outside the contractual requirements (particularly more severe environments or equipment failures than the software was intended to handle); these categories were designated as 1N and 2N respectively. Results of our analysis are shown in Table 3.

Table 3. Analysis of SSA Data

Severity	No. Reports Analyzed (RA)	No.of Rare Reports (RR)	No.of Rare Events(RE)	Ratios		
				RR/RA	RE/RA	RE/RR
1	29	28	49	0.97	1.69	1.75
1N	41	33	71	0.80	1.83	2.15
2	19	12	23	0.63	1.32	1.92
2N	14	11	21	0.79	1.57	1.91
3	100	59	100	0.59	1.37	1.69
4	136	63	92	0.46	0.88	1.46
5	62	25	42	0.40	0.63	1.68
All	385	231	398	0.60	1.23	1.72

Rare events were clearly the leading cause of failures among the most severe failure categories (1 - 2N) and were an important cause among all reports in this population. The number of rare events per report involving rare events (RE/RR, the entries in the last column) remains relatively constant for all severity classes around the average of 1.72. This indicates that inability to handle more than one RE at a time is really at the root of the problem. At this point it is appropriate for each of us to ask ourselves "How often have we traced a thread involving more than one rare event?" and "How often have we concentrated on test cases that involved more than one rare event at a time?" The thoroughness of final testing in the shuttle program surfaced these weaknesses which probably would

have been detected only after they caused operational failures in many other situations.

NASA Fault Tolerant Software Experiment

The importance of multiple rare events as a cause of failure is also borne out in data reported from an experiment in programming redundancy management software [ECKH91]. Twenty versions of a Pascal program were developed at four universities (five versions at each) from the same requirements. The goal of the experiment was to establish the probability of correlated errors for N-version fault tolerant software. The specifications for the program were very carefully prepared and then independently validated to avoid introduction of common causes of failure. Each programming team submitted their program only after they had tested it and were satisfied that it was correct. Then all 20 versions were subjected to an intensive third party test program. The objective of the individual programs was to furnish an orthogonal acceleration vector from a non-orthogonal array of six accelerometers after up to three arbitrary accelerometers had failed. During each of the third-party test runs for which statistics are presented below an accelerometer failure was simulated (there were also runs without simulated failures). The resulting failure statistics presented below were computed from Table 1 of the reference.

Table 4. Failures in Redundancy Management Software

No. of prior anomalies	Observed Failures	Total Tests	Failure Fraction
0	1,268	134,135	0.01
1	12,921	101,151	0.13
2	83,022	143,509	0.58

The number of rare conditions present in a test run was one more than the number listed in the first column (because an accelerometer anomaly was simulated during the test run, and it is assumed that the software failure occurred in response to the anomaly). In slightly over 99% of all tests a single rare event (accelerometer anomaly) could be handled as indicated by the first row of the table. Two rare events produced an increase in the failure fraction by more than a factor of ten, and the majority of test cases involving three rare events resulted in failure.

For those who are daunted by the problems of constructing test cases with three rare conditions, there is one encouraging finding in this article: all versions that failed under three rare conditions had already experienced failures under two rare conditions. Three-quarters of the versions that failed under two (and also under three rare conditions had already experienced at least one failure in the runs represented by the first column. Even more encouraging: programs that had no failures under 0, 1, and 2 rare conditions also did not fail under three rare conditions.

Conclusions and Recommendations

The results reported above all pertain to "final" testing (after all routine tests had been completed and presumably the faults found there had been corrected) or to operation. Thus they represent selected conditions that may not be representative of general software test environments. But this selection permits us to identify fault types that are likely to remain in the program after the conclusion of "normal" testing. Where the nature of a program demands extremely high reliability, it appears therefore that the test scenario should include a substantial number of cases that simulate multiple (at least two) rare conditions.

This can be accomplished in three ways:

- Conventional test case preparation, depending on the skill of the analyst to identify individual rare conditions and to combine them in a meaningful way; this is costly and sensitive to bias (the analyst may select exactly the same conditions that had been anticipated when the software was generated)

- Random testing over a data set that is rich in opportunities for multiple rare conditions. This technique has yielded some encouraging results [BISH88; Sect. 4.10.1][BARN88]. This is economically feasible only where the correct results of a test are easily identified. In the referenced cases back-to-back testing of several versions was used. The generation of suitable data sets is subject to bias but probably to a lesser extent than manual construction of test cases.

- Path testing, particularly where semantic analysis is used to eliminate infeasible paths [HECH90].

The latter technique can be automated and is the only one for which an objective completeness of test criterion can be identified. However, it is costly and will practically be restricted to the most essential portions of a program.

These findings can also be interpreted in a very positive sense of pointing toward a verification of extremely high reliability of programs. Consider the progression of test results shown in the following figure. The change in vertical scale midway along the horizontal axis is introduced to permit a closer observation of what happens in the final stages of a test program that is being conducted along the lines described above. During the initial testing (left side of figure), the large majority of failures occur under routine conditions. Later, as shown in the middle part of the figure, single rare conditions become the predominant cause of failures, and finally multiple rare conditions take over (this is the situation exemplified by Table 3 for the high severity failures).

Assume that it is desired to demonstrate to demonstrate that the probability of failure is less than 10^{-6} per hour. To accomplish this by conventional methods is nearly impossible (e. g,, 1000 computers running for 1000 hours each without failure). However, if it can be shown that all failures observed during the final stages of test are due to at least two independent rare conditions, and if the frequency of occurrence of these conditions is indeed rare, then a good case can be made for showing this program to have at least the right order of reliability. As a practical example, consider a redundancy

management program which developed an anomaly under the following conditions: (1) a hardware failure expected to occur no more frequently than once per 1000 hours, and (2) interruption of the main power source during the recovery from the hardware failure. Again, let us assume that the probability of such an interruption is less than once per 1000 hours. Since the causes of hardware failure and power interruption are assumed to be uncorrelated, the probability of the joint event is less than 10^{-6} per hour.

REFERENCES

BARN88 M. Barnes et al., *Software Testing and Evaluation Methods,* OECD Halden Reactor Project, May 1988

BISH88 P. G. Bishop, editor, *Dependability of Critical Computer Systems 3, Techniques Directory,* Elsevier Applied Science, London & New York, 1988

CAST81 X. Castillo and D. P. Siewiorek, "Workload, performance, and reliability of digital computing systems", *Digest, FTCS-11],* pp. 84-89, IEEE cat. no. 81CH1600-6, June 1981

ECKH91 D. E. Eckhardt, A. K. Caglayan, J. C. Knight, et al., "An experimental evaluation of software redundancy as a strategy for improving reliability", *IEEE Trans. Software Engineering,* vol 17 no 7, July 1991, pp. 692 - 702

HECH90 M. Hecht, K. S. Tso and S. Hochhauser, "The Enhanced Condition Table Methodology for Verification of Critical Software in Ada and C", *Proc. 7th International Conference on Testing Computer Software,* June 1990

IYER82 R. K. Iyer and D. J. Rosetti, "A statistical load dependency model for CPU errors at SLAC", *12th International Symposium on Fault Tolerant Computing,* IEEE Cat. 82CH1760-8, pp. 363-372, June 1982

KANO87 K. Kanoun and T. Sabourin, "Software Dependability of a Telephone Switching System", *Digest, FTCS - 17,* June 1987, pp. 236 - 241

ROSS82 D. J. Rossetti and R. K. Iyer, "Software Related Failures in the IBM 3081: A Relationship with System Utilization", Stanford University, Center for Reliable Computing, CRC Technical Report 82-8, July 1982

VELA83 Paola Velardi and R. K. Iyer, "A Study of Software Failures and Recovery in the MVS Operating System", Stanford University, Center for Reliable Computing, CRC-TR83-7, July 1983

VELA84 P. Velardi and R.K. Iyer, "A Study of Software Failures and Recovery in the MVS Operating System", *IEEE Trans. Computers, Special Issue on Fault tolerant Computing,* Vol. C-33, No. 7, July 1984