NASA Contractor Report 145371

MODELING OF A LATENT FAULT DETECTOR IN A DIGITAL SYSTEM

PHYLLIS M. NAGEL

VOUGHT CORPORATION
HAMPTON TECHNICAL CENTER
HAMPTON, VA 23665

CONTRACT NAS1-13500
SEPTEMBER 1978

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23665

MODELING OF A LATENT FAULT DETECTOR IN A DIGITAL SYSTEM

By Phyllis M. Nagel
Vought Corporation Hampton Technical Center

ABSTRACT

Methods of modeling the detection time or latency period of a hardware fault in a digital system are proposed that explain how a computer detects faults in a computational mode. The objectives were to study how software reacts to a fault, to account for as many variables as possible affecting detection and to forecast a given program's detecting ability prior to computation. A series of experiments was conducted on a small emulated microprocessor with fault injection capability. Results indicate that the detecting capability of a program largely depends on the instruction subset used during computation and the frequency of its use and has little direct dependence on such variables as fault mode, number set, degree of branching and program length. A model is discussed which employs an analog with balls in an urn to explain the rate of which subsequent repetitions of an instruction or instruction set detect a given fault.

INTRODUCTION

The concept of coverage as an important variable in the reliability assessment of fault tolerant computer systems has long been recognized (ref. 1, 2). Coverage, in effect, provides a measure of the chances of a system's recovery in response to a hardware fault. The determination of coverage for a given system largely depends on two variables: The time to detection or latency time of a fault during which the computer continues its computational task undisturbed, and the reconfiguration time, given detection, during which the computer must isolate the fault and implement the recovery strategy of the system. Of the two the latter is the easiest to understand and is the most intuitive to the system designer and consequently is easier to model realistically in reliability calculations. It is no surprise, however, that realistic models of detection time are difficult to find. The variable is highly dynamic, not only fault dependent, as is reconfiguration time, but also dependent on the type and scheduling of the detectors detecting the fault and on the computational burden of the entire system.

Models of coverage usually model the time to detection, in terms of a function of one or more random variables reflecting the characteristics of the detector or detectors used in sensing the fault. CARE II (ref. 3, 4) contains, by far, the most careful development of the mathematical interaction of these variables by introducing the concept of competing detectors on fault classes. Unfortunately the use of this model in assessing the reliability of a specific system is handicapped by the complete lack of data with which to model and forecast the behavior of a given detector beyond the realm of the educated guess.

To rectify this deficiency and understand more completely the nature of latent faults, the present research has selected the comparator/voter for detailed consideration. The importance of this detector is undeniable in that the use of voting across two or more channels as a detector of faulty output is basic to the design of every redundant, reconfigurable computer system. The problem of evaluating the comparator/voter as a detector is not an easy one however, in that it is not one of evaluating the efficiency and performance of a particular piece of hardware. Since the result of a vote is based on the output of a program, the entity being evaluated is the capacity of a program to detect hardware faults in a computational mode.

The importance of a computation-based analysis in the calculation of coverage was recognized by Never (ref. 5). There he argues that if reliability calculations are to reflect the computational needs of the user in the definition of system success then computation-based measures do so more accurately than standard, structure-based analysis. Thus, the present investigation concentrates on computation-based detection of hardware faults in an effort to gain further insight into the interaction between structure and task as it influences coverage.

The working hypothesis governing the experiments presented in this paper is that different programs with varying program features such as the degree of branching, the number of instructions executed the type of instructions executed, the number set used in computation, etc., vary both in their capacity to detect and in their rate of detection. What then does this detection capability depend on for a given program and can it be forecasted prior to computation from physical features of the program itself?

The existence of a program implies the existence of a system within which it is to operate and to accurately evaluate these questions the program should be investigated as it performs in its computational environment. Since large diagnostic general purpose emulators do not exist, the properties of software detection were explored under less dynamic conditions and the

results that follow are relative to this less than realistic environment. The experiments were conducted on a diagnostic emulator with fault injection capability at the gate level currently under development by the Aircraft Electronic Systems Branch at NASA, Langley. The emulator was programmed to emulate a very simple processor with thirteen instructions referred to in this paper as the very simple processor or VSP. Programs were written with this instruction set, run in the presence of randomly injected gate faults and data collected on the accuracy of the output.

EXPERIMENT

The instruction set for the very simple processor contains the following thirteen instructions:

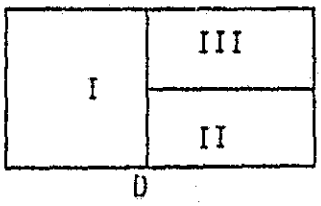
*fetch and *store
*add and *subtract
shift right and shift left
AND and OR
indirect addressing
overflow indicator
*branch
copy to and from temporary storage.

Six programs were written in the language and are described below. The results of the analysis that follows were based on the output of simulating the first five programs and the output of the sixth was used as a confirmation case. As a control device all six programs were coded using only the five starred instruction.

1. Fibonacci (FIB) - Creates a sequence such that any member is the sum of the preceding two members starting with a pair of random initial values. Eight members of the sequence are generated.
2. Fetch and Store (F&S) - Fetches a number from memory and stores

it in another location. This process is repeated eight times.

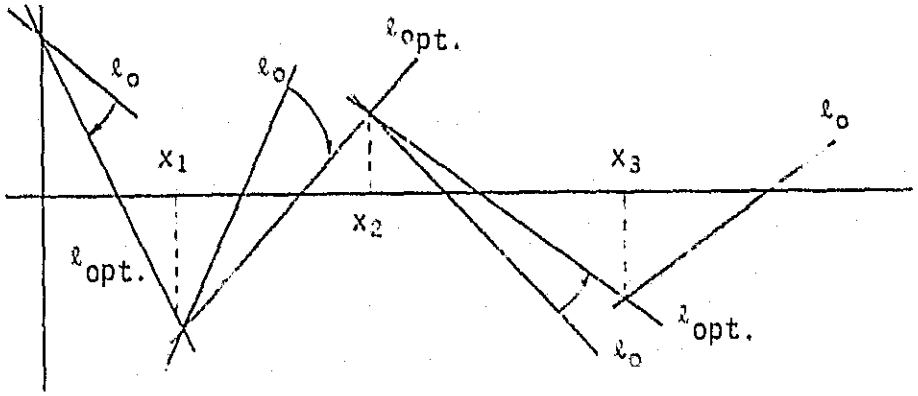
3. Add and Subtract (A&S) - Four subtractions and four sums are computed alternately from values in memory.
4. Search and Compute (S&C) - Two random numbers are chosen from a list of the first twenty numbers and identified by a search. The 20 X 20 square region is divided vertically and horizontally by a random division D and additional computations performed in the areas indicated to form three separate branches:



Branch I - a simple count
Branch II - a subtraction and a count
Branch III- a subtraction and a multiplication

A correct run was determined by a correct identification and a correct branch computation.

5. Linear Convergence (LC) - A line with a random slope and intercept is adjusted in slope so as to cross the x-axis prior to a predetermined x value, x_1. Once crossed, its deviation from the x-axis at x_1 is minimized over slope. From the point on the line just optimized at x_1 a new line of opposite slope is obtained by repeating this process at new value x_2. Iterations are continued until a given number of computer cycles has elapsed. A successful run was one that completed this number of cycles without error.



SAMPLE OUTPUT - LC PROGRAM

6. Quadratic (QUAD) - Computes the value of various quadratic polynomials of the type $Ax^2 - Bx - C$ where A, B and C are positive integers and $-10 \leq X \leq 10$. For a given run, four sets of the four initial values are selected at random and four computations performed.

A program was simulated by running it N times with random input, each run in the presence of a different single gate fault selected at random uniformly over the gate list. For each run the fault was injected prior to computation and the fault mode was determined by treating input and output faults, stuck-at-1 and stuck-at-0 as equally likely alternatives.

It was not the intent of this investigation to explore faults in the voter/comparator but to evaluate how software reacts when executing in the presence of a hardware fault. Thus comparisons were made not by voting over two or more copies but by comparing the output of the simulation to a correct value achieved either by hand calculation or by a fault free run of the same program under the same initial conditions. A record was then kept of the number of runs having faulty output for sample sizes that varied from 97 to 211. The table below contains a summary of the recorded results.

Program	Sample Size	Detections	Estimated Detection Probability	Estimated Standard Deviation
FIB	211	98	.464	.034
F&S	118	42	.356	.044
A&S	208	117	.563	.034
S&C	118	64	.542	.046
LC	133	78	.586	.043
QUAD	97	55	.577	.050

Simulation Results

Note the imprecision in the estimates of detection probability as measured by the standard deviation. This suggests that in general deviations from the given estimates by at least four in the second digit are still quite likely.

Initially a sample size of approximately 100 runs per program was selected as optimal with regard to time and budget. Later two of the programs FIB and A&S, were extended to approximately twice that amount in order to evaluate to some extent the effect of sample size on the stability of the estimate of detection probability.

DATA ANALYSIS

Once the detection probabilities were obtained it was anticipated that variations between them could be explained in terms of variations in program features such as the number of executed instructions, the number of different instructions used in computation, the degree of branching, the number set, etc. The significant features could then be used to predict the performance of a given program's capacity to detect hardware faults in the processor.

The first table of the two that follow contains a breakdown of several program features for each of the first five programs and the second summarizes the instruction set utilized during computation.

Program	No. of Program Statements	No. of Executed Statements	Memory Locations	No. Size	No. of Branches
FIB	12	33	17	0 to 85	0
F&S	18	16	27	0 to 85	0
A&S	22	20	35	0 to 85	0
S&C	344	151.5 (Avg.)	375	-50 to 200	3
LC	378	Random	341	±200	Random

Program Features

Program	Fetch	Store	Branch	Add	Subtract
FIB	X	X	X	X	
F&S	X	X			
A&S	X	X		X	X
S&C	X	X	X	X	X
LC	X	X	X	X	X

Instruction Set

Using linear regression, variations in the probability of detection were explored as a function of the entries for the first four programs in the table of Program Features. Since every combination of variables considered by these methods produced at least one negative regression coefficient, this data did not begin to explain the source of variation in the results of the simulations.

In contrast when the information in the Instruction Set table was investigated a more consistent signal emerged. The following sections explore the nature of this signal and consider the question of the dependence of the probability of detection on the individual variables of the feature table in more detail.

Instruction Set

The primary difference in instruction set between the FIB program and the F&S program is the add instruction. With the addition of the add instruction in FIB to the fetch and store instructions in F&S the corresponding probability of detection jumped from .356 to .464. Similarly when a subtract instruction was added in A&S to the instruction in FIB the probability jumped from .464 to .567 and remained approximately at this level (.542) when the instruction set was held constant in S&C.

To determine if these differences are real or due to statistical error several statistical tests of significance were conducted. First a test for equality between the detection probability of FIB and that of F&S was rejected. Thus the addition of the add instruction in FIB to those of F&S increased the detection probability by a significant amount. Similarly a test for equality between the detection probability of F&S and A&S was rejected so that the inclusion of the subtract instruction increased the detection capability significantly. Since S&C and A&S use the same instruction set, a test for equality between their detection probabilities should not reject if the overall thesis that detection primarily depends on the instruction set is valid. This was indeed the case. The estimated probabilities for these two programs do differ of course but the results of the test indicate that if there are real differences they are still buried in statistical error and therefore are much smaller than the differences between A&S and FIB for example.

Prior to simulating the LC program the early results from A&S were combined with those from S&C to form the estimate

$$\frac{138}{243} = .568.$$

This was used as a point estimate to forecast the behavior of the LC program since the instruction set for LC is approximately the same as that for A&S and S&C. Acknowledging that considerable error still existed in this forecast due to sample size, a 90% confidence interval was computed as an interval estimate. Thus the interval

$$[.568 \pm 1.65\ (.032)] = [.515,\ .621]$$

was the actual forecast for the detection probability of the LC program. The results of the LC program based on 133 runs show a detection probability of .586 which is well within the prediction interval. Using all subsequent runs of A&S changes the interval to

$$[.558 \pm 1.65\ (.0275)] = [.512,\ 603]$$

which still validates the prediction.

If the toy microprocessor should be employed as a serious computational device it would now be possible to forecast the detection probability of any program utilizing all or part of this instruction set of five instructions. The least squares estimate of that part of the detection probability due to the fetch and store instructions, which includes as well the effect of those faults whose detection is common to all instructions, is .356. The additional contribution to the probability due to the inclusion of an add is .108 and the additional due to a subtract is .100. A total point estimate for a program utilizing all five instructions is .564

In the preceding discussion the role of the branch instruction was not established as its effect is not clear. Statistically this instruction has zero effect as a predictor of detection probability. It seems quite plausible that the effect of this instruction over and above the other four instructions to which it is very closely related is so small that it cannot be separated from the random fluctuation still present in the data at these sample sizes.

Fault Mode

Of interest in studying the properties of software fault detection is a determination of the strength of the relationship between detection and fault mode. That is are input faults more detectable than output, or are s-a-1 faults more easily detected than s-a-0? Statistical chi-square tests of independence in contingency tables were performed on all programs for detection versus I/O, detection versus s-a-1, s-a-0 and detection versus the combined signal. None of the s-a-1, s-a-0 tests were significant nor were any of the test conducted on the combined signal. Two of the I/O tests rejected independence namely FIB and S&C, the rest did not. Of the two that implied dependence the dependence was in opposite directions, that is output faults were more likely to be detected by the FIB program than input faults and for S&C the effect was reversed.

When the samples from all five programs were combined the dependence in I/O was no longer significant. All tests were conducted at the 5% level of significance.

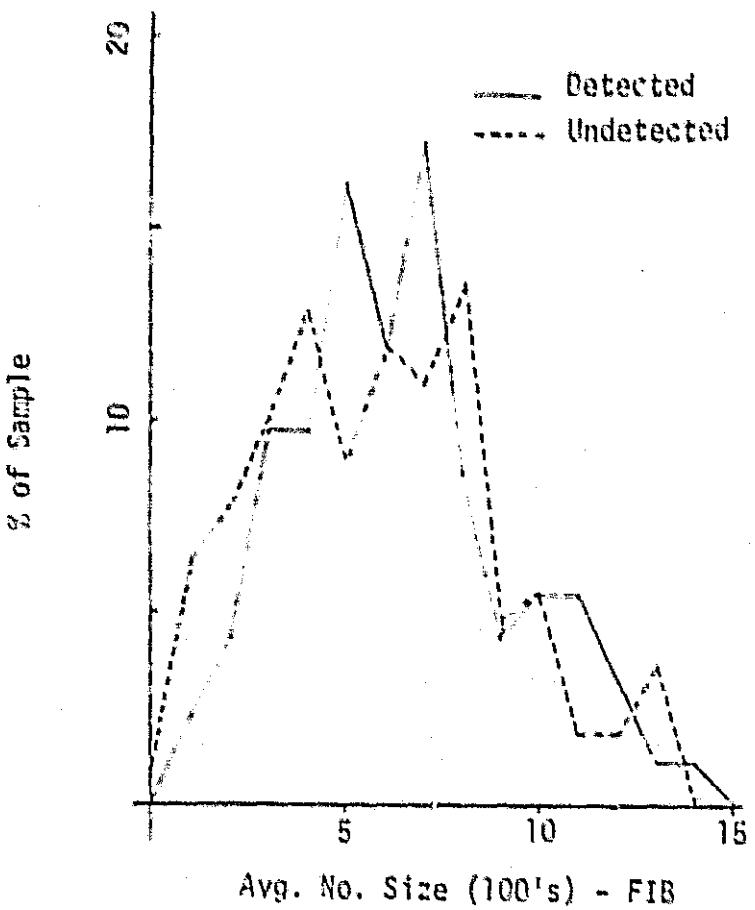
Number Size

The dependence or independence of detection on number size is difficult to measure directly from these experiments in that number size is not a control variable and consequently expands, expands and contracts or changes at random throughout the program. By selectively sampling the data, however, new experiments can be defined which provide some information on the nature of this relationship. These experiments were conducted on the three essentially repetitive programs FIB, F&S and A&S.

The F&S program performed each iteration on an independent random number having full octal range ($0\text{-}8^5$). The experiment on number size consisted of matching a run that detected the fault at random with a run containing a fault that was not detected. For each pair the numbers executing at the time detection occurred were recorded for the detected run and its undetecting companion. Run by run differences across the pairs were then computed and this set of differences formed the basic sample. This same procedure was followed for the A&S program except that for each iteration two independent numbers are involved in the computation instead of one. Thus the differences were taken between the averages of the two numbers executing at the time of detection for the detected run and its paired partner.

It was then hypothesized that if number size was a significant factor in detection the mean value of these differences should shift away from zero. A statistical "t" test was conducted on this mean for each program testing for zero versus non zero. Neither test showed a significant difference from zero. It can therefore be concluded that there is no significant difference in runs that were detected and those that were not with respect to number size.

For FIB, number size is a function of the size of the two random initial inputs and therefore a slightly different experiment was defined. First the averages were sorted by whether detection occurred, creating a sample from each of two populations. A test was conducted for equality of the two means versus inequality as again it was hypothesized that if detection depended on number size there should be a shift away from equality. The test showed no significant difference between them. The density for each sample is plotted below.



Branching and Program Length

The S&C program contains 3 distinct branches that can be ranked short, medium and long as the average number of executed statements is 42.3, 113.9 and 305.3 respectively. This provides a means of testing if the branch effect is significant. The detected and nondetected runs were categorized by branch to form a 2X3 contingency table. A test for independence was conducted and failed to reject thereby implying that there is no strong evidence supporting the hypothesis that detection depends on branch in this case.

Another test was conducted on this data to see if there was any dependence of detection on the number of statements actually executed during the running of the program. The number of executed statements ranged from 14 to 432. The data was divided into 100's and a probability of detection calculated for each. The results are given below.

No. of Statements	No. of Samples	No. Detected	Probability
0 - 99	56	27	.482
100 - 199	22	15	.682
200 - 299	15	11	.733
300 - 399	19	10	.526

Program Length vs. Detection - S&C Program

The four data points over 399 were omitted from consideration. It was hypothesized that if there was no dependence of detection on the number of executed statements these probabilities should all be statistically equal. A chi-square test conducted on these four proportions assuming equality against all alternatives failed to reject. Thus at these sample sizes there is no evidence to support the contention that the detection is dependent on

the number of executed statements. This is further substantiated when it is noted that the data from 200 on is all from branch III so that even though computational discrepancy is minimal, the proportions, though different, are contrary to expectations.

When other programs are considered with respect to the number of statements executed the same confusion is evident. The F&S program takes about half as many executions as FIB to detect .8 as many faults. A&S, on the other hand, takes about .7 as many executions as FIB to detect 1.2 times as many faults.

Detection Time

The signal coming from the number of statements executed is confounded with the signal coming from the nature of the statements being executed. Thus it may be more reasonable to treat the entire program as an entity and attempt to predict its performance as a whole.

When proportion of detection is plotted against the number of times the program has repeated itself at the time of detection it is apparent that there is a dependence. The three programs A&S, FIB and F&S are repetitive or nearly repetitive and provide a means for evaluating the nature of this dependency. The following table gives the number of failures for each of these programs as a function of which repetition the program was executing when detection occurred.

	AAS		FTR		FAS	
Repetition	Freq.	% of Total	Freq.	% of Total	Freq.	% of Total
1	65	.313	55	.261	33	.182
2	20	.096	12	.057	6	.051
3	14	.067	10	.047	2	.017
4	5	.024	2	.009	2	.017
5	3	.014	0	.000	2	.017
6	3	.014	1	.033	5	.043
7	4	.019	4	.019		
	117	.563	98	.464	47	.356
n	208		211		118	

Detection as a Function of the Number of Looks

Several models have been investigated in an attempt to characterize the efficiency of detection with regard to repeated looks at a fault. Only the two of them discussed below appear to adequately explain the behavior displayed in the above table from both a numerical and intuitive point of view. The first is based on a model proposed in CARE II (ref. 3, 4) that time to detect, for the comparator/voter operating continuously, is exponentially distributed with a constant multiplier denoting the overall probability of detection. The second utilizes an analogy of balls in an urn to model the way in which a program detects a fault where generating a fault is equivalent to reaching into the urn and marking a ball. Still another phenomenon related to this general question is cited in the section on the confirmation program.

Exponential Model

Modifying the assumptions of the CARE II model slightly to reflect that the programs were terminated after only eight iterations, leads to the following structure on the density function of

$$y = \min(t, T)$$

where t is the time of detection measured in repetitions and T is the truncation time of the test, in this case eight:

$$f(y) = \begin{cases} p_0 \lambda e^{-\lambda y} & y < T \\ p_0 e^{-\lambda T} + Q_0 & y = T \\ 0 \text{ elsewhere} & \end{cases} \qquad (Q_0 = 1 - p_0)$$

There are two reasons why a fault may not be detected under the assumptions of this model: one, because the fault may not be detectable by the program in question (denoted by α) or two, because sufficient time has not elapsed for the fault to be seen (denoted by β). Q_0 measures α, the probability of a fault remaining undetected for all time, $p_0 e^{-\lambda T}$ measures β, the probability of going undetected because of insufficient time or attention by the program, and p_0 is the stand alone detection probability.

Maximum likelihood estimators for the parameters p_0 and λ of this model reduce to solving the following pair of simultaneous transcendental equations:

$$p_0 (1 - e^{-\lambda T}) = n/N$$

$$\lambda [\Sigma t_i + \frac{n T e^{-\lambda T}}{(1 - e^{-\lambda T})}] = n$$

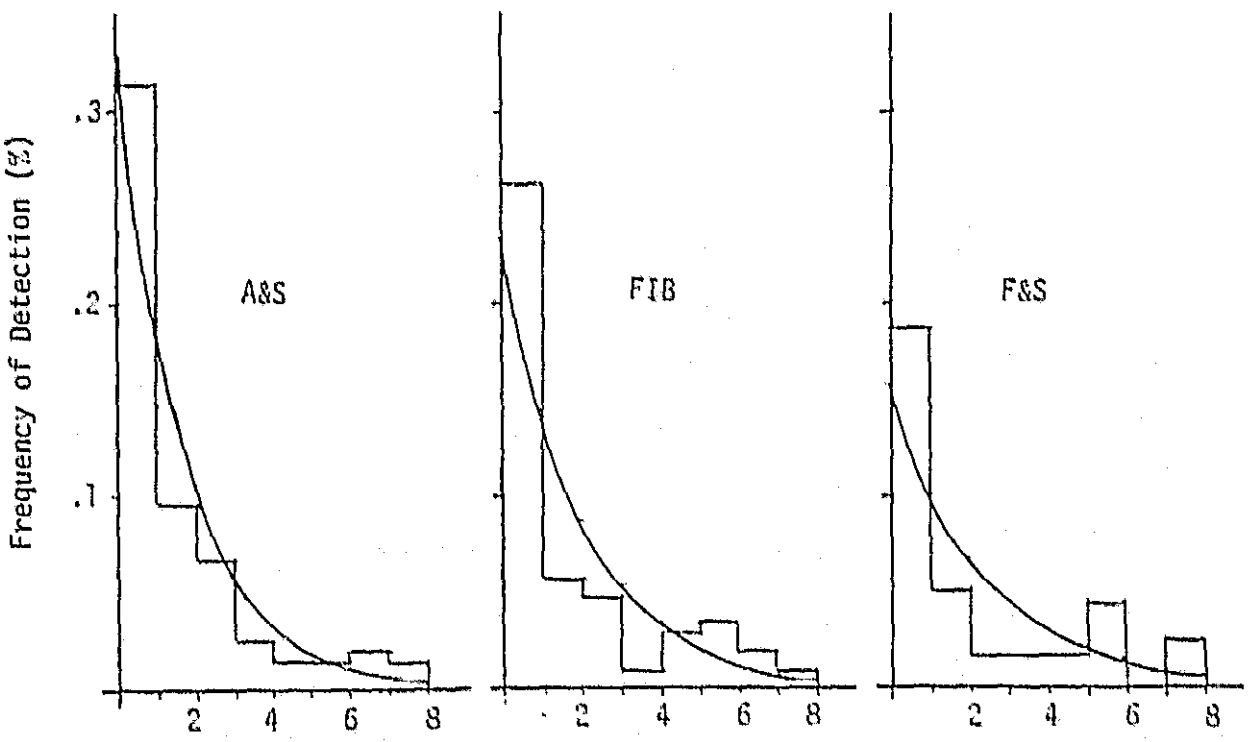
where n = number of runs terminating in a detection,
N = number of runs
and t_i = detection time in terms of repetitions.

By solving the second equation iteratively for λ and substituting into the first the following table was computed.

Program	P_o	λ	λ/P_o	α	β
A&S	.568	.577	1.02	.432	.006
FIB	.474	.491	1.04	.526	.009
F&S	.371	.398	1.07	.629	.015

MLE Estimates - Exponential Model

It is interesting to note that even though there is wide variation in the individu estimates of P_o and λ there is remarkable stability in their ratio. The final two columns give estimates for the probability that the fault will remain permanently undetected by the program and the probability that the fault is not detected due to test truncation, respectively. Plots of this function superimposed on histograms of the detection data for each of the three programs appear below.



Exponential Model

Urn Model

Though the previous model explains much of the variation in detection time, it only provides information on the rate of detection and not on the mechanism of detection. For this reason a new model is proposed in this section which explores, by means of an analogy with balls in an urn, the question of what a program experiences while it is executing in a faulty processor and, once determined, provides a method of forecasting a program's detection efficiency as a function of time.

Let S be the set of all gate states for a given processor. For this example then, S is the set of all triplets of the form (x_1, x_2, x_3) where x_1 is the gate, x_2 designates its use as an input or an output gate and x_3 denotes its value. Let A be the subset of all gate states in S that are encountered during repeated use of a given program. If, by analogy, the set S is a set of balls in an urn consisting of two colors, say red and blue, representing the sets A and $\bar{A}$, respectively, then generating a random fault is equivalent to reaching into the urn and marking a ball. The probability that the ball is red is simply the stand alone probability of detection for that program designated P_0 in the exponential model.

To complete the analogy it will be assumed that executing a program is equivalent to reaching into the urn and withdrawing a handful of red balls, since the program only exercises gate states in its detectable set. Each successive repetition of the program is modeled by assuming that the withdrawn balls are replaced in the urn after every draw, and the sizes of the draws are statistically the same. Instead of being entirely independent, however, the draws share some balls in common. That is, from repetition to repetition the assumption is that some gate states are common to every computation performed on that program and some are not.

These assumptions lead to the following model for the probability of detecting a fault on the kth draw, P_k:

$$P_1 = PP_0 \qquad k = 1$$

$$P_k = (1 - P)(1 - r)^{k-2} P_0 \qquad k > 1$$

where P_0 is as before, the stand alone probability of detection, P is the probability of detection on the first draw given that the draws are from the red balls, and ρ represents a probabilistic measure of the unshared gate states. Thus the probability of detection on the first draw is the probability of withdrawing the marked ball given that the ball is red, times the probability that a red ball is marked. The probability that the marked ball is withdrawn on the second draw is $(1-P)_\rho P_0$, where the $(1-P)_\rho$ factor now represents the conditional probability that the marked ball is not withdrawn on the first draw, times the probability that it is selected on the second in that part of the draw unshared by the first, given that the marked ball is red.

Modifying this distribution slightly to account for the fact that the test was truncated at k=8, the distribution for y, where y=min (k,8) and k is the time of observed detection, becomes

$$h(y) = \begin{cases} P\,P_0 & y = 1 \\ (1-P)(1-\rho)^{y-2}\,P_0 & 2 \le y \le 8 \\ P_0 \sum_{i=8}^{\infty} (1-P)(1-\rho)^{i-2}\,\rho + Q_0 & y = 8 \quad Q_0 = 1-P_0 \\ 0 & \text{elsewhere} \end{cases}$$

By assuming that $P_0 \sum_{i}^{\infty} (1-P)(1-\rho)^{i-2}\rho$ is zero, approximate MLE estimates for P_0, P and ρ can be obtained for each of the three programs A&S, FIB and F&S by solving the equations

$$P_0 = n/N$$

$$P = n_1/n$$

$$\rho = \frac{n-n_1}{\Sigma t_i - m}$$

where n, N and t_i are defined as in the exponential model and

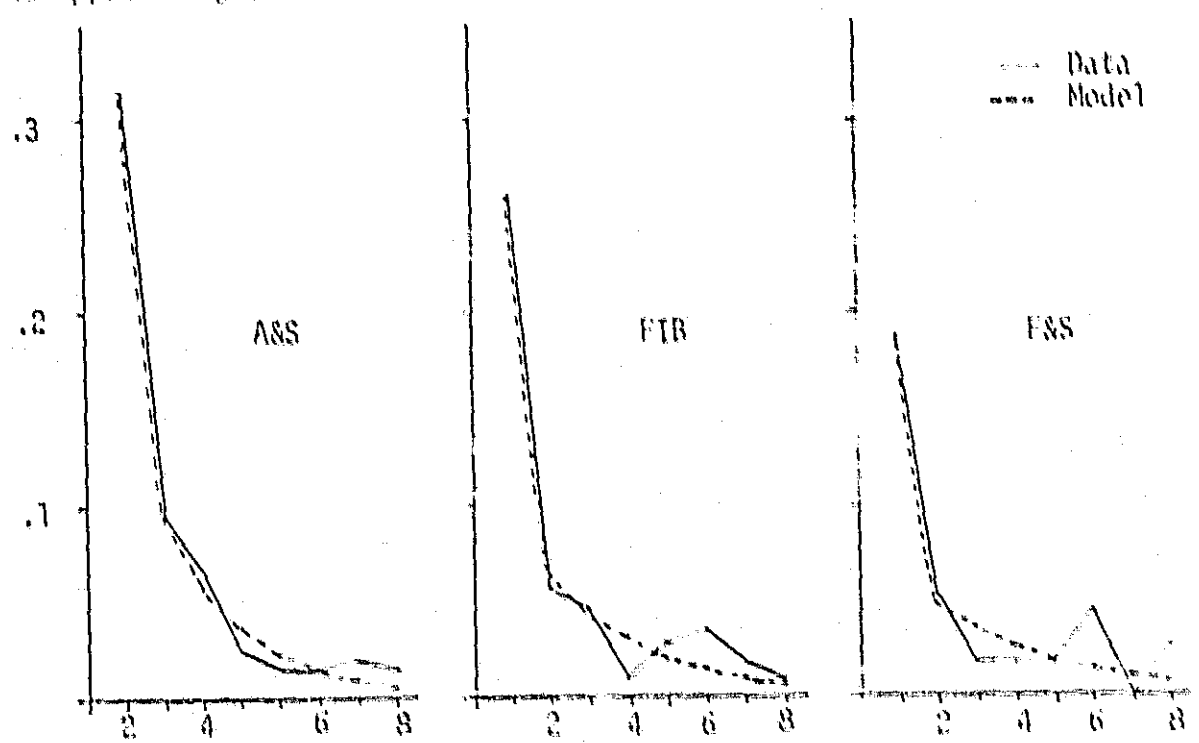
n_1 = number of runs terminating in a detection during the initial repetition.

Results of these computations are found in the table below:

	P_0	P	ν	α	β
A&S	.563	.556	.385	.437	.010
FIB	.465	.561	.319	.535	.015
F&S	.356	.534	.286	.644	.033

M.L. Estimates - Urn Model

Plots of this model appear as dashed curves in the graphs that follow. Though the function and the observed data plotted with it are both discrete, the points have been connected with straight lines for better visability. It should be noted that the estimate for P_0 forces the model to coincide with the data at the first repetition. Since, too, there are three free parameters to estimate in this model instead of only two in the exponential model, the fit is appreciably better.



Urn Model

The quality of the fit, however particularly with regard to explaining the fall off at t = 2 is consistent and appears to provide an explanation for detection not contained in the exponential model.

It is premature to predict the performance of the parameters of the model in any definitive way except to note certain broad features. In particular it is interesting that P is a fairly stable parameter over all three programs and that the variation between programs, in addition to that already discussed earlier with regard to the differences in overall performance as measured by P_o, is also evident in ρ. In particular A&S and FIB are the two programs which have been run at sample sizes large enough to nearly stabilize the estimate of P_o. For these programs

$$\frac{P_o(A\&S)}{P_o(FIB)} = \frac{\rho(A\&S)}{\rho(FIB)} = 1.21$$

That is the ρ's are in the same ratio as the P_o's correct to two decimal places. This fact is not true, however, when these programs are compared to F&S and attributing this to the remaining instability in the estimates of F&S can only be conjectured.

Confirmation Program

After data on the first five programs had been collected and a preliminary model formulated, it became apparent that the data for the S&C and LC programs did not entirely support the model with regard to rate of detection. If the model on rate was correct, subsequent repetitions of these programs should detect more faults and thus increase their overall probability of detection. At the same time, however, the estimated overall probability of detection was already at the predicted level based on the first repetition of these programs only and hence under this prediction subsequent repetitions should detect very little. Since, too, both of these programs are much more complicated than those previously analysed, it is conceivable that neither model is correct and that further explanation is in order. Because of restrictions on the number of

cycles for the VSP, it was not practical to extend either of these two programs to more repetitions. Thus, to resolve this apparent contradiction a sixth program QUAD was written and simulated as an example of a more complicated program running in repetition.

Since the QUAD program uses all of the five previously defined instructions, the data from the A&S, S&C and LC programs were combined to form a point estimate of .564 as a forecast for the behavior of QUAD. The overall detection probability of QUAD based on a sample size of 97 is .57. Not only is this value close to the predicted value but it is also close to the point estimate .563 for A&S though A&S is a highly time dependent detector. When the data for QUAD was sorted by repetition, 52 of the 56 detected runs detected on the first repetition, three detected on the third and one detected on the fourth based on a test truncation time of 4. That is, almost all of the faults causing faulty output did so the first time the program looked at the fault; a phenomenon confirming previous results obtained from the LC and S&C programs. Thus, though the overall level of detection is predictable, there is little or no dependence of detection on future looks at the same fault when programs get complicated.

Though at first thought these results are surprising, they become less so when the complexity of the programs are studied with the assumptions of the rate model in mind, that is, when they are interpreted by the same model they appear to violate. A structure common to the S&C, LC and QUAD program is their repetitive use of comparison and multiplication loops to do much of the computation. Since both of these instruction sets involve the use of all five of the instructions under consideration, their use is repetitive but inside the program. As a result the cummulative effect of detection is sensed by the program but is only tapped after many repetitions for each of the instructions have been excercised. Thus it is the integral that is observed and not the repetition. In effect, this leads to a new model of detection based not only on the instruction set but on the frequency with which a given instruction is used during computation recognizing that detection is a decaying function of frequency but with nearly constant rate.

Suggestions for Continued Experimentation

Several issues have been raised in the preceding discussion that could be validated somewhat more completely if additional experimentation was conducted. Whether they should be continued on the VSP or await a more realistic computer model is difficult to assess in that it depends on the credibility of the small computer. In any event the following list contains suggestions for continued experimentation.

1. The prediction method for predicting stand alone probability is severely hampered due to the statistical error in the estimated probability. Therefore several programs need to be run to large sample sizes in order to sharpen the forecasting tool.
2. Forecasting the parameters of the urn model requires new programs run to larger sample sizes as suggested in 1. These would provide data for predicting the level and stability of P and the dependence of ρ on P_0.
3. The interactive model for more complex programs involving the dependence of detection on instruction set and their frequency of use during computation is not definitive. A series of controlled experiments could be designed to investigate this dependence.
4. When the interactive model is completed it could be tested for its prediction capability by varying the fault injection time.
5. The emulator now has a memory with fault injection capability. Though it is expected that the degree of modeling complexity is no way near as complicated as that for the processor, simulations could be conducted to determine if this is so.
6. When a processor is used as a component in a fault tolerant flight computer, tasks will be run back to back

with repetitions of one program interspersed with repetitions of others. As a result it is not sufficient to study a program in isolation but in tandem with other programs. As a first step this issue could be explored using two programs in tandem that have already been studied in isolation.

7. Recent information has indicated that pin failure instead of gate failure is a more realistic model of IC failure. Some of these experiments should be repeated to explore the consequences of this information.

CONCLUSION

The intent of this investigation has been to propose methods of modeling the duration and extent of latent faults by exploring the way a computer detects faults in a computational mode. In particular the desire was to account for as many variables as possible affecting detection and to use them in forecasting a given program's detecting ability prior to computation.

This investigation has shown that relative to the simplified version of a microprocessor used in these experiments, detecting capacity of a program largely depends on the instruction subset used in computation and the frequency of its use and has little dependence, if any, on such variables as fault mode, number size, branching and program length.

For simple programs that control the use of a given instruction or instruction set, two models are explored that show a decaying dependence of detection on repetitive use of the program. The most interesting of the two models employs a simple analogy with balls in an urn, to explain both the mechanism of detection and the rate at which subsequent repetitions of the program detect a given fault. For more complicated programs, the data supports the contention that where repetitive use of an instruction is excercised during computation, detection is equivalent to sensing the cummulative effect of repeated exposure to the fault. Thus complex programs detect more the first time they see a fault and repeated looks provide little additional information.

Though these results should be regarded as exploratory in part because of the computational environment and in part because there is still a considerable amount of statistical error in the data, nevertheless they should be considered with some seriousness. The picture they provide of the computing process is both reasonable and self consistent and it is conceivable that with a larger facility emulating a realistic processor, the methods and results presented here can provide insight into a more general theory of computation based detection.

Vought Corporation Hampton Technical Center
Hampton, Virginia 23666
June 15, 1978

REFERENCES

1. Bouricius, W. G.; Carter, W. C.; and Schneider, P. R.: Reliability Modeling Techniques for Self-Repairing Computer Systems. Proc. ACM Annual Conf., pp 295-309, 1969.

2. Bavuso, Salvatore J.: Impact of Coverage on the Reliability of a Fault Tolerant Computer. NASA TN D-7938, 1975.

3. Raytheon Co., Equipment Development Laboratory: Reliability Model Derivation of a Fault-Tolerant, Dual, Spare-Switching, Digital Computer System. NASA CR-132441, 1974.

4. Raytheon Co., Equipment Development Laboratory: An Engineering Treatise on the CARE II Dual Mode and Coverage Model. NASA CR-144993, 1976.

5. Meyer, John: Computation-Based Reliability Analysis. IEEE Trans. on Computers, vol. C-25, No. 6, 1976, pp 578-584.

1 Report No	2 Government Accession No.	3 Recipient's Catalog No
NASA CR-145371		

4 Title and Subtitle

Modeling of a Latent Fault Detector in a Digital System

5 Report Date
September 1978

6 Performing Organization Code

7 Author(s)

Phyllis M. Nagel

8 Performing Organization Report No

10 Work Unit No

9 Performing Organization Name and Address

Vought Corporation
Hampton Technical Center
Hampton, Virginia

11 Contract or Grant No
NAS1-13500

13 Type of Report and Period Covered
Contractor Report

12 Sponsoring Agency Name and Address

National Aeronautics and Space Administration
Washington, DC 20546

14 Sponsoring Agency Code

15 Supplementary Notes

NASA Technical Manager: S. J. Bavuso

16 Abstract

Methods of modeling the detection time or latency period of a hardware fault in a digital system are proposed that explain how a computer detects faults in a computational mode. The objectives were to study how software reacts to a fault, to account for as many variables as possible affecting detection and to forecast a given program's detecting ability prior to computation. A series of experiments was conducted on a small emulated microprocessor with fault injection capability. Results indicate that the detecting capability of a program largely depends on the instruction subset used during computation and the frequency of its use and has little direct dependence on such variables as fault mode, number set, degree of branching and program length. A model is discussed which employs an analog with balls in an urn to explain the rate of which subsequent repetitions of an instruction or instruction set detect a given fault.

17 Key Words (Suggested by Author(s))	18 Distribution Statement
Latent Fault reliability Fault Coverage Detection	Unclassified - Unlimited

19 Security Classif. (of this report)	20 Security Classif. (of this page)	21 No. of Pages	22 Price*
unclassified	unclassified	26	

* For sale by the National Technical Information Service, Springfield, Virginia 22161