JOURNAL OF AEROSPACE COMPUTING, INFORMATION, AND COMMUNICATION
Vol. 7, October 2010

Software Development Queuing Model

Norman Schneidewind*
Naval Postgraduate School, Pebble Beach, California 93953

DOI: 10.2514/1.40133

Queueing models have been applied to the fault detection and correction process [1,2]. In addition, analysis of the time spent by faults in a software testing system has led naturally to an increased interest in the dynamics of queuing networks, which are used to model such systems. Analytical as well as simulation models have been used to study the behavior of queues and fault correction stations in testing systems [3]. Both analytical and simulation models are employed in an attack on relieving fault bottlenecks in fault correction systems.

I. Introduction

WE all know about people queuing up to check out at a grocery store. Perhaps less obvious is the concept of faults that have been detected in during software testing queuing up to be corrected. Testing is an important software development function. Interestingly, testing is essentially a queuing process. When a fault is in a queue, it attempts to move to a fault correction station (see Fig. 1). If, at that moment, the station is full, the fault is forced to reside in the queue until a place becomes available in a station. The queue remains blocked during this period of time [4]. Therefore, it may not be possible to test, detect faults, and correct faults immediately because there could be queue of faults that are being corrected; therefore, the faults must be queued and wait their turn for service. Therefore, queuing models are employed to estimate quantities such as wait time, service time, and number of faults waiting for service.

There are two ways that queues can be analyzed. One way is by analytical models whose solutions represent steady state or mean value solutions. These models also assume a single queue feeding multiple fault correction stations as shown in Fig. 1. This is akin to banks in which a single queue feeds multiple teller stations. Another approach is simulation that is similar to the approach of discrete-event simulation [5] that is used to analyze reliability of component-based software. This approach relies on random generation of faults in components using a procedure that computes the interfailure arrival time of a faults into queues [6]. Simulation has the advantage of providing finer grain solutions of, for example, number of faults in individual fault correction stations. In this approach, as shown in Fig. 2, multiple queues feed multiple stations. We use this configuration because the NASA Shuttle software testing process involves multiple testers detecting faults that form multiple queues $1, \ldots, i, \ldots, n$ and, for efficiency purposes, streams of faults are fed from the queues into multiple correction stations $1, \ldots, c, \ldots, m$ (i.e., fault correction specialists and computers).

We use both approaches in this paper and compare results. The analytical approach use the classical models [7]. For the simulation approach, we wrote a C++ program. Results will differ because the simulation model uses random number tests to determine when an event, such as a fault entering a queue, would occur, whereas the analytical model does not deal in events. Rather, it computes expected values, such as the expected number of faults in a queue.

Received 29 July 2008; accepted for publication 20 October 2009.
* Fellow of the IEEE, IEEE Congressional Fellow, US Senate 2005, ieeelife@yahoo.com

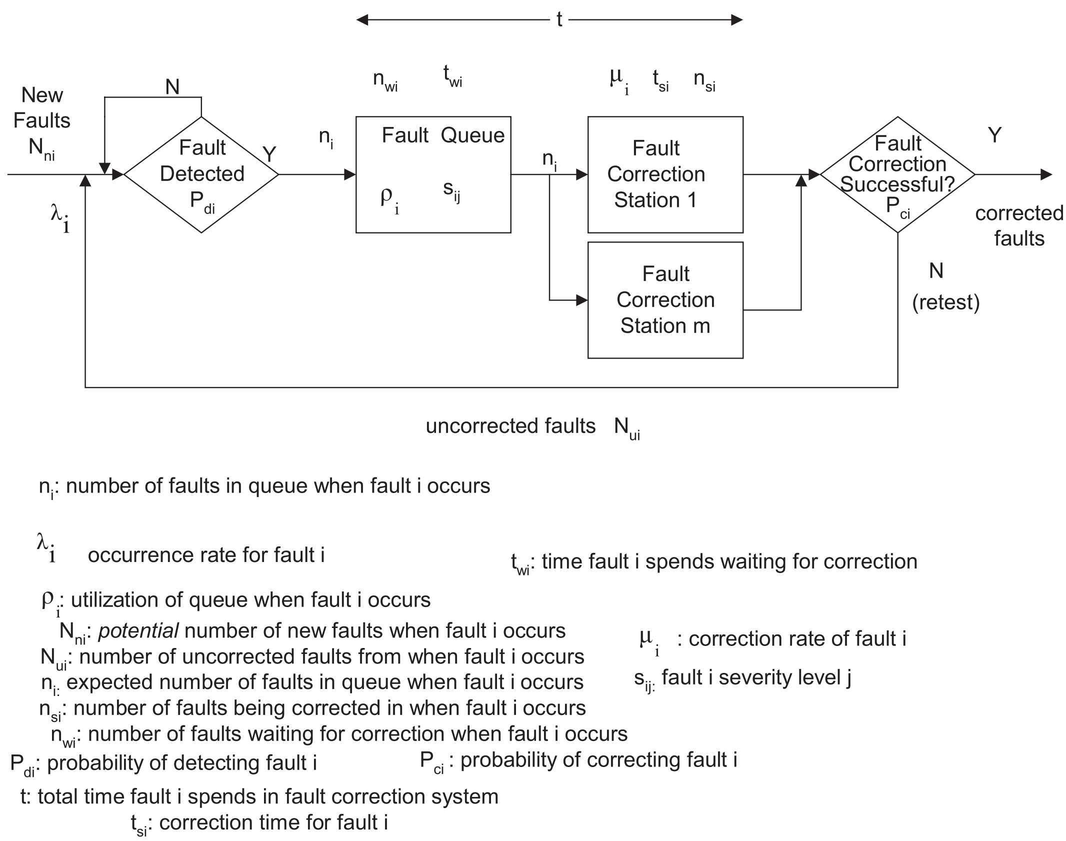

Fig. 1 Analytical model queueing process.

Objectives

One objective is to identify the optimal number of fault correction stations, where "optimal" is defined as the number of stations where additional stations would yield diminishing returns in correcting faults. A second objective is to identify which faults may require excessive processing time. This is done in order to feed back this information to improve the development process. Another objective is to compare assignment of faults to correction stations in *order of occurrence* with assignment based on choosing the station with *minimum existing fault count*. The first plan corresponds to the situation where we must attempt to correct faults without delay. The second plan corresponds to batching the faults for the purpose of achieving fault correction efficiency. In order to implement this plan, the faults counts are sorted in ascending sequence and assigned to the first empty station.

II. Queuing Models for Software Development

As an example of a fault detection and correction system, we model multiple fault correction stations (i.e., servers) using fault data from NASA Space Shuttle release OI4 (operational increment). As opposed to classical queuing models that are restricted to using steady state or mean values, we model the queue discipline as individual faults occur. In order to determine whether the faults occur according to an exponential distribution, we conducted a *t*-test of the actual times between fault occurrences vs an exponentially distributed set derived from its mean. The probability of the two sets coming from the same population is 0.9288.

A. Analytical Model

1. Fault Generation

We see in Fig. 1 that a stream of new faults N_{ni} attempt to enter the fault correction system. Whether they will be "admitted" depends on whether they will be detected. We generate this quantity by using the simulation program.

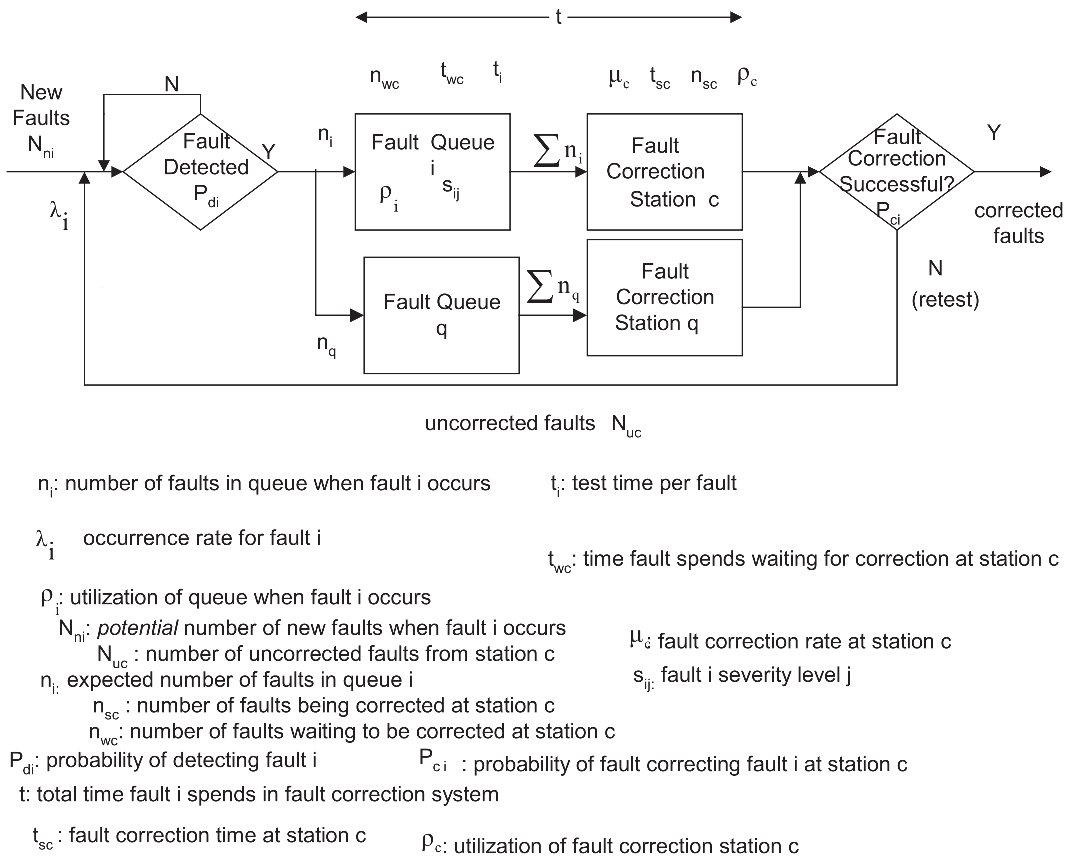

Fig. 2 Simulation model queuing process.

The analytical model is supported by the simulation model to the extent that a stream of faults attempting to enter the system are subjected to a detection test by comparing P_{di}, generated by random numbers, with test values, also generated by random numbers. If the test succeeds, N_{ni} is incremented by "1"; otherwise, no incrementation takes place.

2. *Fault Severity*

Queuing theory has been applied to correcting software maintenance problems. The mean time to correct faults is a useful process metric. The most severe faults can preempt less severe faults. Given the reasonable assumption that the most severe faults have the highest impact on reliability, reducing correction time is given priority attention [8]. Therefore, it is appropriate to consider the severity of faults when analyzing a fault correction system because the time to correct faults, and the number of faults at correction stations, is a function of their severity. Furthermore, increased correction times caused by high severity impact the time that faults wait to be corrected. This is analogous to the supermarket when customers with complex purchases cause large processing times at the checkout counter causing long lines and waiting times of other customers. To reflect this condition, we compute a fault severity weighting factor w_{ij} in equation (1) and apply it to *increase* the appropriate quantities below. Of course these quantities do not actually increase, but the weighting scheme is a method to more realistically represent the fault queue process. Note that for these severities, s_{i1} is the most severe and s_{im} is the least severe, but s_{i1} has the lowest numerical value and s_m has the highest value. Given these conditions, the weights in equation (1) sum to 1.0 over the m severity levels.

$$w_{ij} = \frac{(1 - (s_{ij}/s_{im}))}{2} \tag{1}$$

3. Faults in Queue

Once the number of new faults N_{ni} has been generated, the weighted expected number of faults n_i in the queue when fault i occurs in Fig. 1 is determined by whether the new faults are detected, as shown in Eq. (1a) as follows:

$$n_i = (N_{ni} * P_{di})(1 + w_{ij}) \quad \text{(1a)}$$

4. Fault Occurrence Rate

The occurrence rate of fault i is given by the reciprocal of the time between fault i and fault $i+1$ occurring in Eq. (2) as follows:

$$\lambda_i = \frac{1}{t_{ai}} \quad \text{(2)}$$

5. Uncorrected Faults

The expected number of uncorrected faults after faults have been processed at stations is computed by noting in Fig. 1 that some of the n_i faults do not get corrected. Thus, accounting for probability of fault correction, P_{ci}, we have

$$N_{ui} = (n_i) * (1 - P_{ci}) \quad \text{(3)}$$

6. Queue Utilization

Now we compute queue utilization, a critical parameter that is the probability of the queue being busy, or the utilization of the queue when fault i occurs, noting that to compute utilization, we must increase the occurrence of faults n_i by the fault severity weight.

$$\rho_i = \frac{n_i}{\sum_{i=1}^{q} n_i} \quad \text{(4)}$$

7. Queue Dwell Times

The total time that faults spend in fault correction system is equal to the total number of faults in the queue divided by the fault input rate, increased by the fault severity factor as follows:

$$t_i = \frac{n_i}{\lambda_i} \quad \text{(5)}$$

The time that a fault spends being corrected increases with queue utilization ρ_i, for given number of stations c and time between fault occurrences t_{ai}, and is computed in Eq. (6) as follows:

$$t_{si} = \rho_i (c t_{ai}) \quad \text{(6)}$$

The fault wait time is computed in Eq. (7) by subtracting the time that faults spend being corrected, computed in Eq. (6), from the total time that faults spend in the fault correction system, computed in Eq. (5) as follows:

$$t_{wi} = t_i - t_{si} \quad \text{(7)}$$

8. Queue Correction Counts

Using the time faults spend being corrected from Eq. (6) and the fault occurrence rate, the number of faults being corrected is computed in Eq. (8) as follows:

$$n_{si} = \lambda_i t_{si} \quad \text{(8)}$$

9. Queue Correction Rates

Using Eq. (6), we can compute the rate of fault correction in Eq. (9) as follows:

$$\mu_i = \frac{1}{t_{si}} \quad \text{(9)}$$

10. State of the System

If there are zero faults in the system, this is a good omen for fault i because it can be processed for correction without delay. Thus we would like to know the probability of the state of the system. The probability of zero faults in the fault correction system when fault i occurs is computed in Eq. (10) [7] as follows:

$$p_{0i} = \frac{1}{D_i} \tag{10}$$

where

$$D_i = \sum_{n_i=0}^{c} \left(\frac{(c\rho)^{n_i}}{n_i!} + \frac{(cp)^c}{c!(1-\rho)} \right) \tag{11}$$

Then the probability of $n_i \geqslant 0$ faults in the fault correction system is given by Eq. (12) as follows:

$$p_{ni} = 1 - p_{0i} \tag{12}$$

An important concern in software testing is: what is the probability that a fault will end up being queued [9]? Another way of phrasing this question is: what is the probability of n_i faults already in the queue when fault i is detected? If ni is too large, faults are blocked [10], and deferred for later processing. Using Eq. (10), we compute this probability in Eq. (12a) as follows:

$$P_{ni} = p_{0i} \left(\frac{(\lambda_i/\mu_i)^{n_i}}{n_j!} \right) \tag{12a}$$

11. Wait Queue Counts

Since, in general, faults cannot be corrected immediately, we use Eq. (10) to compute the number waiting in the queue for correction as follows:

$$n_{wi} = \lambda_i t_{wi} \tag{13}$$

12. Efficiency of Testing

The efficiency of testing fault i is computed as follows, where N_{ui} is the number of uncorrected faults and t is the time spent in the fault correction system:

$$E_i = \frac{n_i - N_{ui}}{t_i} \tag{14}$$

B. Simulation Model

Simulation can be considered as a tightly coupled and iterative three-staged process comprised of model design, model execution, and execution analysis [11]. Our model design is represented in Fig. 2. Model execution is represented by Fig. 3 and by the equations below. Execution analysis is the reporting of simulation results in Sec. III.

As in the case of the analytical model, it is necessary to account for effect of fault severity by weighting the number of faults in queue i, n_i, and the number of faults in correction station, n_c, by the factor $(1 + w_{ij})$. Thus, borrowing Eq. (11) from the analytical model, the number of faults in queue i, n_i, are weighted in Eq. (14a). The weighting factors are computed by Eq. (1). Since the faults n_c are the summation of faults n_i, as shown in Fig. 3, these faults will have been weighted.

$$n_i = (N_{ni} * P_{di})(1 + w_{ij}) \tag{14a}$$

1. Test Time, Time of Fault Occurrence, and Fault Occurrence Rate

Through testing, NASA faults occur at times T_i (see Fig. 2) [12], where n_c represents the number of faults that are assigned to fault correction station c. Since n_c faults are in station c at time T_i, the test time per fault is computed as follows:

$$t_i = \frac{T_i}{n_c} \tag{15}$$

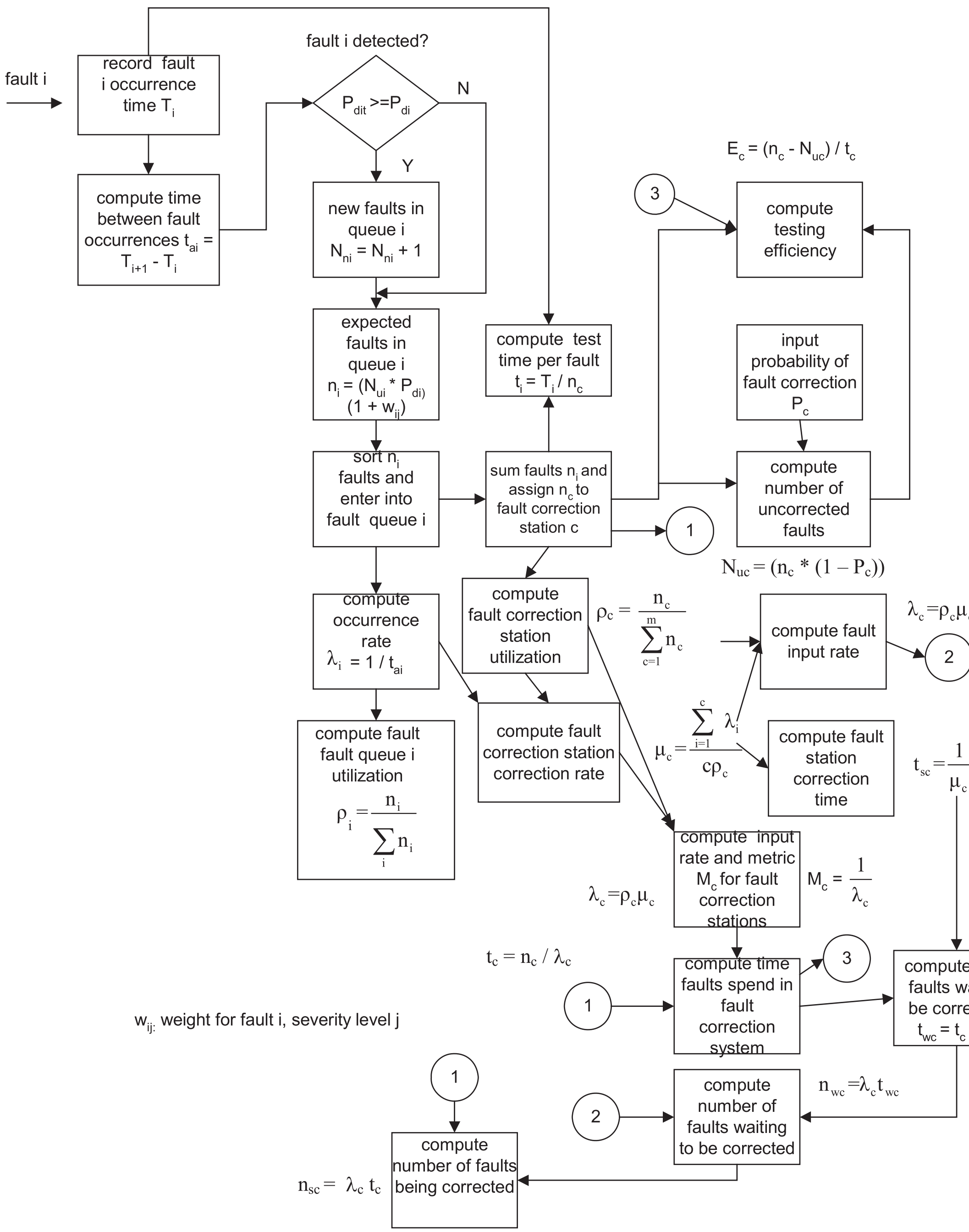

Fig. 3 Simulation model queuing computations.

We are interested in computing the occurrence rate of i faults, λ_i, and comparing it with the fault correction rate μ_i to see whether the correction rate can keep up with the occurrence rate. If it cannot, this means that the fault correction system becomes unstable. First, we need the time between consecutive fault occurrences, t_{ai}, computed in Eq. (16) as follows:

$$t_{ai} = T_{i+1} - T_i \tag{16}$$

Then using Eq. (16), the fault occurrence rate is computed as follows:

$$\lambda_i = \frac{1}{t_{ai}} \tag{17}$$

2. Queue Utilization

In the simulation model we compute the weighted utilization for each queue. This approach gives a more realistic assessment of utilization than is the case with the analytical model that computes a utilization for a single queue. The approach is initiated by detecting a fault in Fig. 2, incrementing the number of new faults N_{ni}, computing the number of faults n_i in queue i, and assigning them to fault correction stations. This is accomplished in Fig. 2 and in the C++ simulation program by sorting the n_i faults in ascending order, summing the faults in pairs of adjacent queues, and assigning the pairs sequentially from 1 to m, where m is the number of stations. The utilization of each fault queue i is computed as follows:

$$\rho_i = \frac{n_i}{\sum_{i=1}^{q} n_i} \tag{18}$$

3. Fault Correction Station Utilization

In addition to queue utilization, it is important to compute the utilization of each fault correction station because, after all, this is where the primary action in software testing takes place. Therefore, the weighted utilization of fault correction station c is computed in Eq. (19) where n_c is the number of faults assigned to station c and m is the number of stations. Note that the denominator in Eq. (18) is equal to the denominator in Eq. (19) because all of the faults are assigned to the stations.

$$\rho_c = \frac{n_c}{\sum_{c=1}^{m} n_c} \tag{19}$$

4. Fault Correction Rate and Time

In order to compute the station fault correction rate μ_c, we must sum the fault occurrence rates in the queues λ_i, to arrive at a fault occurrence rate for fault correction station c, as shown in Eq. (20) as follows:

$$\mu_c = \frac{\sum_{i=1}^{c} \lambda_i}{c\rho_c} \tag{20}$$

Once fault station correction rate has been computed in Eq. (20), we can compute the fault station correction time t_{sc} in Eq. (21) as follows:

$$t_{sc} = \frac{1}{\mu_c} \tag{21}$$

5. Time Spent in System and Input Rate

Now we can compute the time faults spend in the fault correction system (waiting to be corrected and being corrected) t_c by using the rate of fault input to fault correction stations, λ_c, and the number of faults n_c at the stations in Eq. (22).

$$t_c = \frac{n_c}{\lambda_c} \tag{22}$$

Using Eqs. (19) and (20), the input rate to fault correction stations is computed in Eq. (23) as follows:

$$\lambda_c = \rho_c \mu_c \tag{23}$$

6. Wait Time, Number Waiting, and Number Being Corrected

Then, once the fault correction time is computed in Eq. (21), the time that faults have to wait to be corrected t_{wc} is computed by Eq. (24):

$$t_{wc} = t_c - t_{sc} \tag{24}$$

Continuing, with t_{wc} in hand, we can compute the number of faults waiting to be corrected n_{wc} as follows:

$$n_{wc} = \lambda_c t_{wc} \tag{25}$$

Using the time that faults spend being corrected t_{sc} from Eq. (21), we compute the number of faults being corrected in Eq. (26) as follows:

$$n_{sc} = \lambda_c t_{sc} \tag{26}$$

7. Uncorrected Faults

As noted in Fig. 2, not all faults are corrected due to deficiencies in the test process (e.g., inadequate test cases). Therefore we need to estimate the number of faults from station c that were not corrected by multiplying the number of faults in station c by the probability of *not* correcting faults, as shown in Eq. (27) as follows:

$$N_{uc} = n_c * (1 - P_c) \tag{27}$$

8. Testing Efficiency

Testing efficiency at correction station c is computed as follows, where N_{uc} is the number of uncorrected faults and t_c is the time faults spend in station c. The objective is to identify the number of stations that maximizes the ratio of number of *corrected* faults to time spent in the station:

$$E_c = \frac{(n_c - N_{uc})}{t_c} \tag{28}$$

9. Fault Correction Effectiveness Metric

Since waiting time, correction time, and total time spent in the fault correction system are quantities that correspond to *all* faults waiting, being corrected, and total number in system, respectively, it is important to normalize these quantities by their respective number of faults, where M_{wc} is normalized wait time, M_{sc} is normalized correction time, and M_c is normalized total time.

$$M_{wc} = \frac{t_{wc}}{n_{wc}}, \quad M_{sc} = \frac{t_{sc}}{n_{sc}}, \quad M_c = \frac{t_c}{n_c}$$

With some algebra, it can be shown that all three metrics reduce to:

$$M_c = \frac{1}{\lambda_c} \tag{29}$$

the reciprocal of the fault input rate to station c. At first blush, this result seems counterintuitive, but upon reflection, we can see that an increasing input rate lead to efficiency in fault correction because the station is kept busy. This result holds up as long as the station utilization does not become excessive (i.e., $\rho_c > 0.90$). The objective is to identify the number of stations that minimize Eq. (29).

A similar result holds for the analytical model with

$$M_c = \frac{1}{\sum_i^c \lambda_i} \tag{30}$$

pertaining to the metric for station c. The fault occurrence rates are summed in Eq. (30) in order to provide a fair comparison with M_c in the simulation model that uses summed occurrence rates.

III. Model Results

A. Test Time Tradeoffs

Of great concern to software testers is the tradeoff between test effort, represented by test time, and the number of faults removed by the test effort. Figure 4 shows that testing and queuing efficiencies increase with increasing number of fault correction stations. However, increasing the number of fault correction stations may not be feasible [13] because doing so would require more test personnel and computer equipment. Therefore, a practical value of number of fault correction stations in Fig. 4 is $c = 3$. The application of this plot would be to serve as a planning document for subsequent releases of the software, assuming similar testing and fault occurrence characteristics, as is the case with the Shuttle. The reason for the large values of test time and time in the system is that the Shuttle software is tested continuously by the developer, in the simulation testbed, in the Shuttle Simulator for astronaut training, at the launch site, and in flight [14].

B. Optimal Number of Fault Correction Stations, Test Efficiency, and Fault Correction Effectiveness

As stated, one of our objectives is to identify the optimal number of fault correction stations to use in a software testing system, where "optimal" can have several interpretations. One interpretation is the number of fault correction stations whether testing efficiency is maximum. This occurs at $c = 4$ in Fig. 5 for both the analytical and simulation models. Of course, each application would have a different solution, but the type of plot in Fig. 5 serves as a roadmap for *any* application.

A second way of viewing station optimality is to employ fault correction effectiveness, as illustrated in Fig. 6. Here, the solution is the same using the simulation model, as was the case in Fig. 5. However, now the solution is $c = 2$ for the analytical model compared with $c = 4$ in Fig. 5. Now what if the solutions differ, as in this case. The way to solve the dilemma is to choose the fault correction effectiveness criterion if this is a safety critical application, as in the case of the Shuttle. Otherwise, opt for the testing efficiency solution.

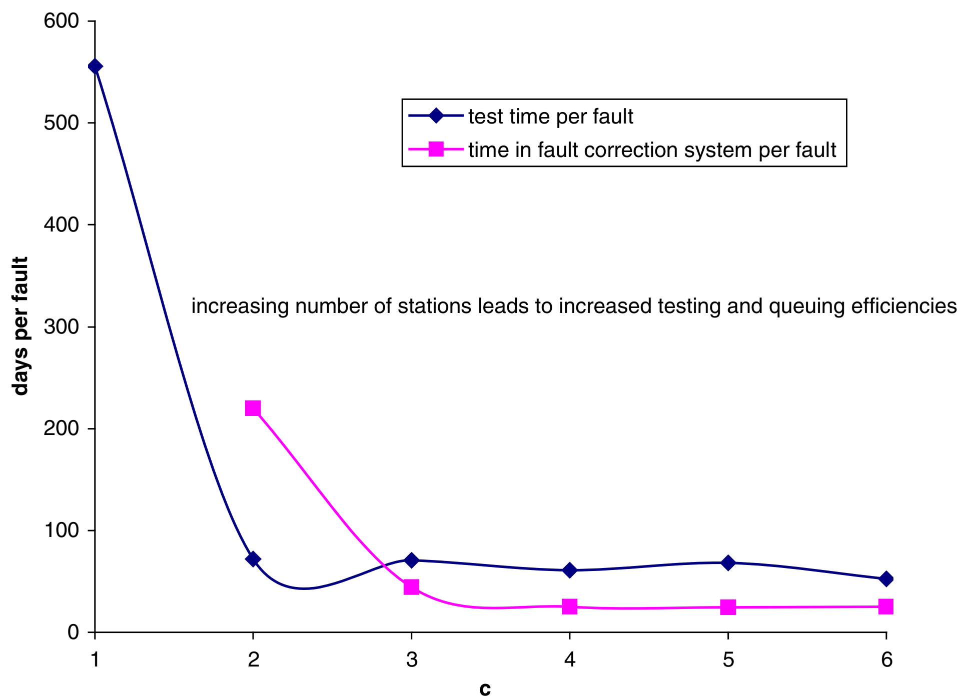

Fig. 4 NASA Space Shuttle OI4: test time per fault and time in system per fault vs number of fault correction stations *c* (simulation model).

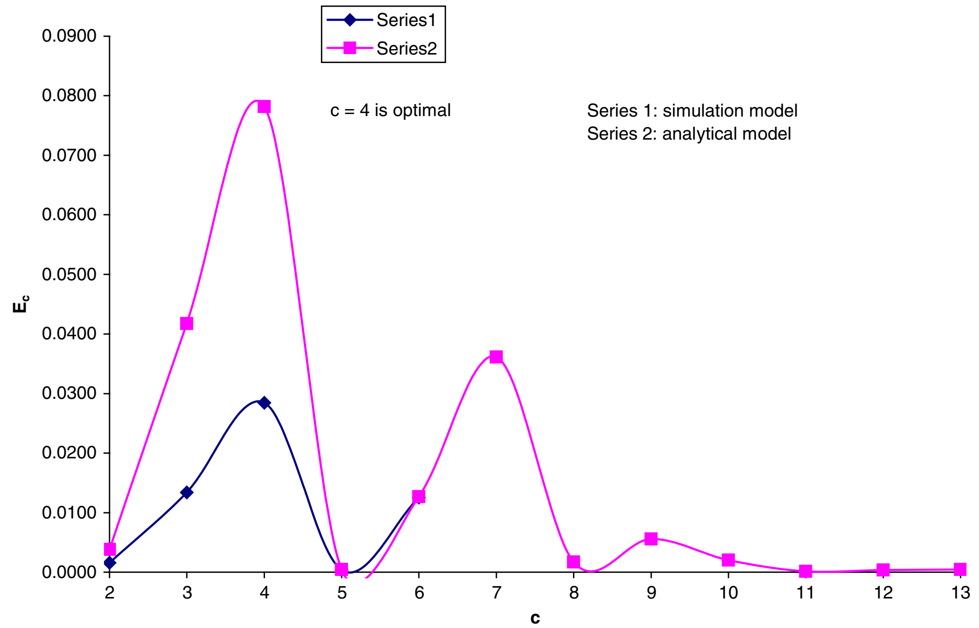

Fig. 5 NASA Space Shuttle OI4: test efficiency E_c vs number of fault correction stations c.

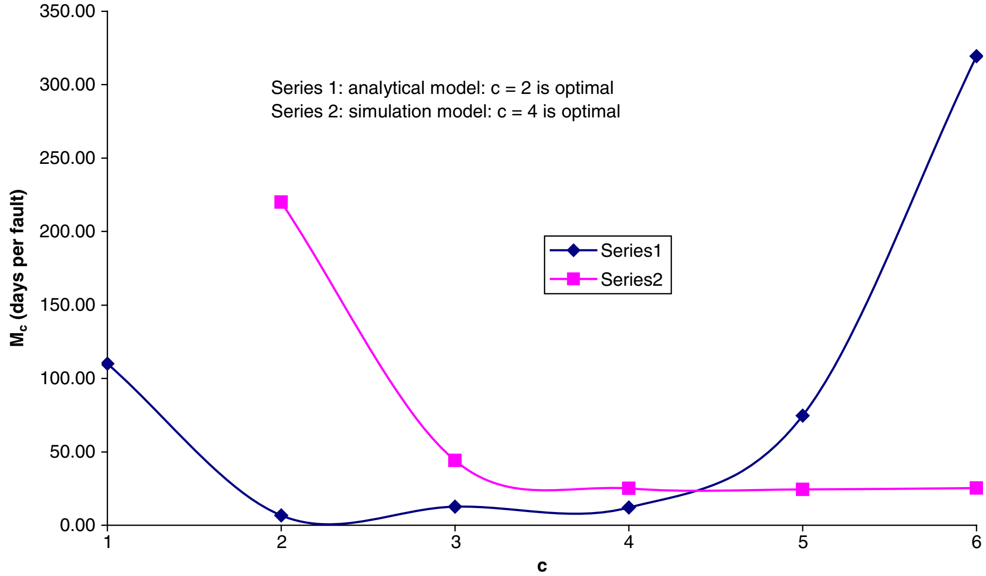

Fig. 6 NASA Space Shuttle OI4: fault correction effectiveness metric M_c vs number of fault correction stations c.

C. Worst Case Number of Fault Correction Stations

Equally important with identifying the optimal number of fault correction stations is the identification of the "worst case" situation. This criterion is based on the number of fault correction stations corresponding to the maximum number of uncorrected faults, as shown in Fig. 7 for the two models. Again, other applications could yield other solutions, but this approach would be applicable to various applications.

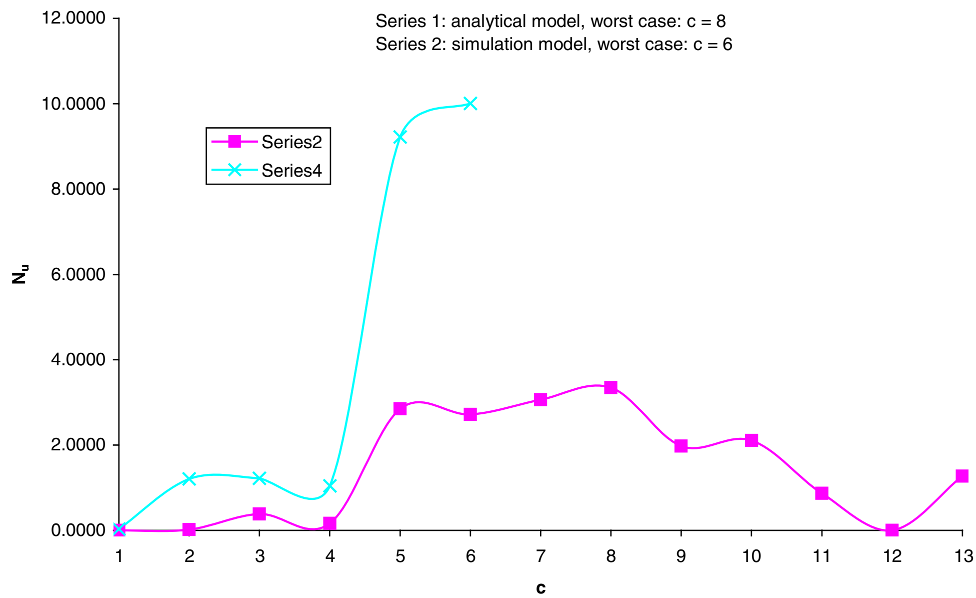

Fig. 7 NASA Space Shuttle OI4: number of uncorrected faults N_u vs number of fault correction stations c.

D. Order of Occurrence Assignment vs Minimum Existing Fault Count Assignment

As described, we use two methods of assigning faults to station as follows 1) order in which faults are detected and 2) examine the existing fault count in stations and assign where the count is minimum. To implement (2), we sort the number of faults and assign them in ascending order. As mentioned, (1) is applicable to safety critical systems where urgency of fault correction is paramount. On the other hand, (2) is applicable where time can be taken to batch

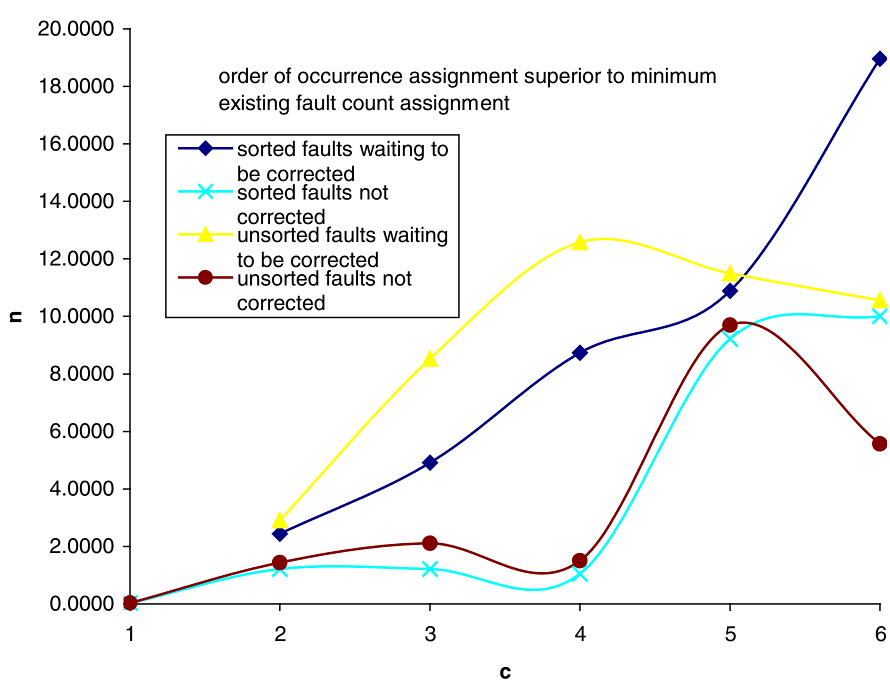

Fig. 8 NASA Space Shuttle OI4: fault counts n vs number of fault correction stations c (simulation model).

the faults in an attempt to achieve test efficiency and fault correction effectiveness. A surprising result is shown in Fig. 8 where (1) is superior to (2) because the sorted faults plot continues to increase, whereas the unsorted faults plot reaches a maximum and then decreases. This is a fortuitous result since we are using Shuttle data.

E. Ability to Queue Faults

We would expect that the ability to queue fault — after they are detected — for subsequent correction in fault correction stations, would increase with increasing fault input into the system. We have this expectation because, with

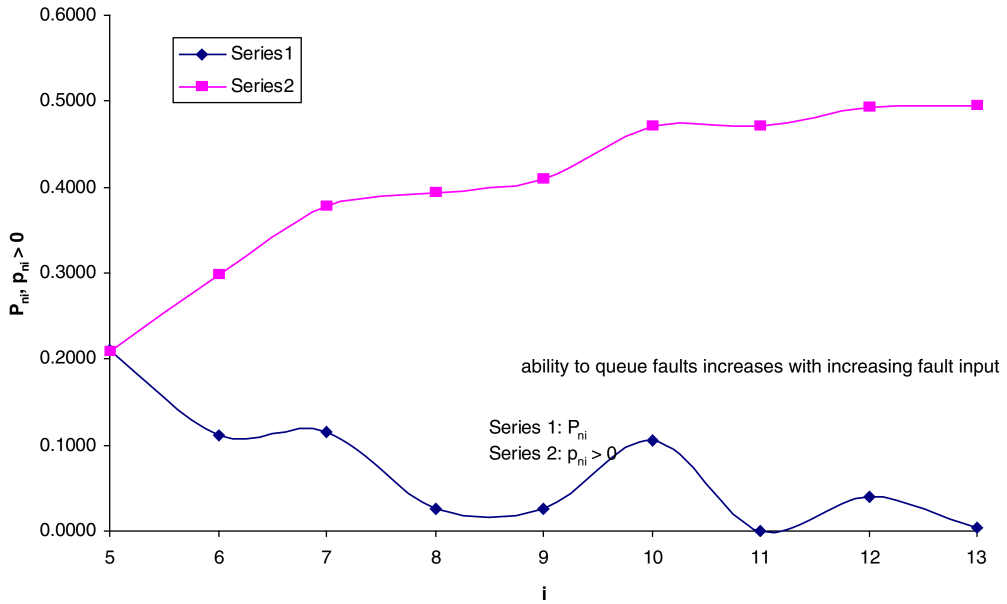

Fig. 9 NASA Space Shuttle OI4: probability of n_i faults in queue P_{ni} and probability of one or more faults in queue $p_{ni} > 0$ vs fault i (analytical model).

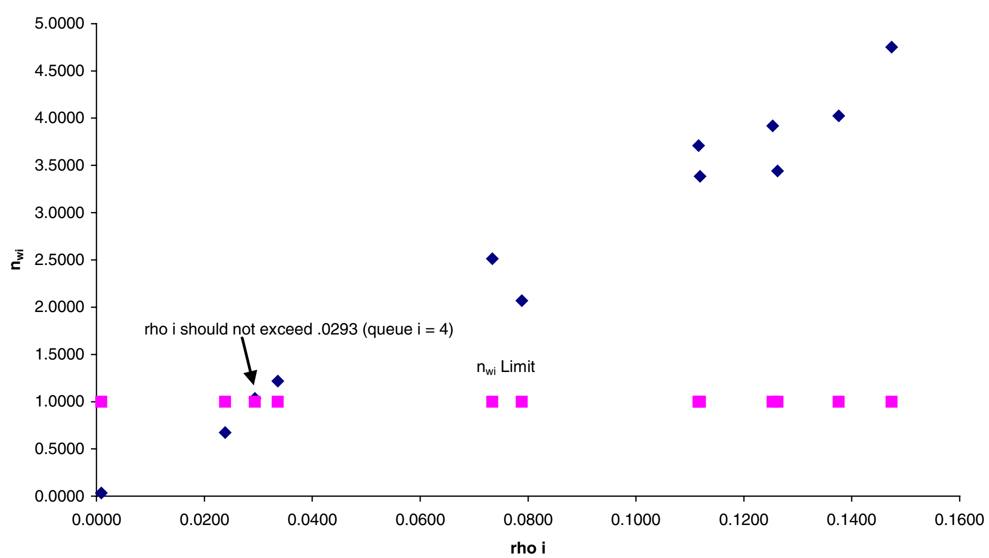

Fig. 10 NASA Space Shuttle OI4: number of faults waiting in queue i n_{wi} vs queue utilization ρ_i.

increasing experience in testing a set of faults from a given application, efficiency in moving faults from detection to queues (see Fig. 1) would increase. Fig. 9 attests to this result.

F. Tracking Number of Faults Waiting as a Function of Queue Utilization

It is important to track a key queuing performance metric, such as the number of faults waiting in queue i to be corrected in a fault correction station, as a function of queue utilization so that the limiting value of utilization can be identified [15]. This is shown in Fig. 10 where the utilization limit is identified based on a desired number of fault waiting not to exceed 1.000. Queues 1–4 satisfy this criterion, but not the remaining queues. Since number of faults waiting is a function of queue input rate, the solution would be to reduce these rates for queues 5, . . . , 13.

IV. Summary

We used analytical and simulation models to evaluate the testing efficiency and fault correction effectiveness of the fault detection and correction process. Two models were used so that we would have a reasonableness check on the solutions, although we must add that the analytical model yields steady state or mean value results and the simulation model provides event-driven values. Thus, we would not expect the results to be identical. Interestingly, Fig. 5 showed identical results. Our objectives as follows: 1) to identify the test time tradeoff function, showing how fault removal varies with test time; 2) to identify the number of fault correction stations that produce the best values of testing efficiency and fault correction effectiveness; 3) to identify the worst case number of stations in terms of the number of uncorrected faults, since uncorrected faults are obviously detrimental to the reliability of the software; and 4) since we were interested in whether assigning faults to stations on the basis of minimum number of existing faults would be beneficial, we evaluated this alternative via simulation. More important than specific numerical results that we obtained is the methodology that we demonstrated that can be applied to all applications.

References

[1] Musa, J. D., Iannino, A., and Okumoto, K., *Software Reliability: Measurement, Prediction, Application*, McGraw-Hill, New York, 1987.

[2] Huang, C.-Y., and Huang, W.-C., "Software Reliability Analysis and Measurement Using Finite and Infinite Server Queueing Models," *IEEE Transactions on Reliability*, Vol. 57, No. 1, March 2008, pp. 192–203.
doi: 10.1109/TR.2007.909777

[3] Manish, K. G., and Michael, C. F., "Queueing Theory in Manufacturing: A Survey," *Journal of Manufacturing Systems*, Vol. 18, No. 3, 1999, pp. 214–240.

[4] Akyildiz, I. F., "On the Exact and Approximate Throughput Analysis of Closed Queuing Networks with Blocking," *IEEE Transactions on Software Engineering*, Vol. 14, No. 1, Jan. 1988, pp. 62–70.
doi: 10.1109/32.4623

[5] Gokhale, S., et al., "Reliability Simulation of Component-Based Software Systems," *Proceedings of the 9th International Symposium. Software Reliability Engineering*, IEEE Publications, Piscataway, NJ, 1998, pp. 192–201.

[6] Wang, X., and Qu, H., "Queuing Analysis and Performance Evaluation of Workflow through WFQN," *Proceedings of the First Joint IEEE/IFIP Symposium on Theoretical Aspects of Software Engineering (TASE'07)*, IEEE Publications, Piscataway, NJ, 2007, pp. 178–187.

[7] Hillier, F. S., and Lieberman, G. J., *Introduction to Operations Research*, 7th ed., McGraw–Hill, New York, 2001.

[8] Gokhale, S. S., and Mullen, R. E., "Queuing Models for Field Defect Resolution Process," *Proceedings of the 17th International Symposium on Software Reliability Engineering (ISSRE'06)*, IEEE Publications, Piscataway, NJ, 2006, pp. 353–362.

[9] Ramaswamy, R., "How to Staff Business-Critical Maintenance Projects," *IEEE Software*, Vol. 17, No. 3, May/June 2000, pp. 90–94.
doi: 10.1109/52.896255

[10] Dai, Y.-S., Pan, Y., and Zou, X., "A Hierarchical Modeling and Analysis for Grid Service Reliability," *IEEE Transactions on Computers*, Vol. 56, No. 5, May 2007, pp. 681–691.
doi: 10.1109/TC.2007.1034

[11] Savino-Vazquez, N.-N., and Puigjaner, R., "A UML-Based Method to Specify the Structural Component of Simulation-Based Queuing Network Performance Models," *Proceedings of the Thirty-Second Annual Simulation Symposium*, Society for Modeling & Simulation International, San Diego, CA, 1999, p. 71.

[12] Schneidewind, N. F., "Reliability Modeling for Safety Critical Software," *IEEE Transactions on Reliability*, Vol. 46, No. 1, March 1997, pp. 88–98.
doi: 10.1109/24.589933

[13] Di Penta, M., Casazza, G., Antoniol, G., and Merlo, E., "Modeling Web Maintenance Centers Through Queue Models," *Proceedings of the Fifth European Conference on Software Maintenance and Reengineering*, 2001, p. 131–138.

[14] Keller, T., and Schneidewind, N. F., "A Successful Application of Software Reliability Engineering for the NASA Space Shuttle," *Proceedings of the Software Reliability Engineering Case Studies, International Symposium on Software Reliability Engineering*, Nov. 3, Albuquerque, NM, Nov. 4, IEEE Publications, Piscataway, NJ, 1997, pp. 71–82.

[15] Cherukuri, N., Kandiraju, G., Gautam, N. and Sivasubramaniam, A. "Queueing Model for Performance Analysis of a Network Interface Card", *Proceedings of the Industrial Engineering Research Conference*, Inst. of Industrial Engineers, Norcross, GA, 2003.

Michael Hinchey
Associate Editor