A Proposal for Standardized Software Dependability Data

Herbert Hecht
SoHaR Incorporated
8421 Wilshire Blvd. Suite 201
Beverly Hills, CA 90211
herb@sohar.com

Introduction

Software that does not fail is essential to many current applications of digital technology, and this need will increase as more functions critical to the individual or the community are being automated. Examples are intelligent highways, use of robots in direct contact with the public, medical monitoring, and drug administration. Based on current specifications for critical software, the maximum tolerable failure rates for the most severe modes range from 10^{-7}/hr for complex multi-computer installations to 10^{-10}/hr for simpler programs running on free-standing computers [FAA91, FAA88]. To demonstrate attainment of such low failure rates requires approaches that go much beyond the established methods of software reliability estimation, e. .g, [MUSA87]. Indeed, a large fraction of the development cost of high integrity systems must be devoted to the assessment of their dependability through software quality assurance, inspection and test. Yet most serious students of the field agree that none of the assessment methods and tools currently employed provides an estimate that has a high correlation with the achieved dependability *for the required high reliability levels.* Because we do not know what and how much the individual steps of the assessment contribute to the validity of the estimate, there is a distinct possibility that some of the effort was wasted. Similar uncertainty exists regarding the value of development methodologies and tools to the attainment of extremely high probability of success. The total development and assessment cost for high dependability systems is so large that even a small reduction of the effort by replacement of ineffective with more effective procedures will result in large economic benefits.

Failures of high dependability systems are (by definition) rare in operation and even during the latter stages of development. In extremely few instances are sufficient failures documented to permit firm conclusions about the effectiveness of individual steps in the development or assessment process. Thus the individual project is not likely to be in a position to identify effective methodologies or tools. One obvious step to correct this difficulty is to compare failure experience of systems that were developed and underwent verification by diverse methodologies. One obvious step for a more effective and rational assessment of dependability is to compare the field failure types and rates (or availability or other dependability characteristics) of systems that underwent verification by diverse methodologies. Obtaining failure data on even a single high integrity system is not a task for the faint-hearted, and failure data *suitable for comparison* from programs that have been developed and verified by a variety of methodologies have to the author's knowledge never been collected. A number of data repositories have been publicized and sometimes even funded, but their effectiveness in providing insights into causes of failures and their prevention has been negligible. The diversity of data formats, and of definitions of such elementary terms as *failure, detection means, and execution time,* have made meaningful comparisons impossible. At least a part of the reason for this discouraging state of affairs has been the volume of information requested in prior data collection efforts. The following is an attempt to arrive at a modest set of data that will still be sufficient for at least a gross comparison of the effectiveness of various dependability assessment methods, and to propose that this data set serve as the starting point for a standards effort.

Proposed Data Structure

This section describes requirements for and implementation of an error classification methodology for high integrity systems. It is largely based on a software error classification scheme recommended for nuclear power safety systems [HECH95]. The emphasis is on the purposes of the classification, the required data, and their logical ordering. There is no intention to propose a specific data format or database management system. Nevertheless, it is very convenient to adopt the terminology of a generic database manager as will be explained now. A classification scheme may be thought of as a table or as a *file*. The columns of the table are the classification topics and are

referred to as *fields* and the rows as *records*. In most instances we have suggested the allowable entries into the fields, and these are referred to as *categories*. Where a given field is subject to hierarchical decomposition, its categories become fields at the next lower level. As an example, the origin field of a fault classification file allows categories of hardware, software, skinware, and environment. At the next level the hardware will be broken down into the specific component that was faulty. At this point "hardware" is a field, and the component names represent the categories. Restricting the entries to defined categories facilitates comparison and makes for a more focused analysis than is possible with the entry of unconstrained text or numeric values.

Several general recommendations have become apparent from the review of prior efforts in classification and these are summarized below.

(a) The classification framework must be open, including at the top level, to accommodate changes in the computing environment, in fault identification techniques, and in recognition of new failure effects. In general there should be an "other" category for each field. In the following it is assumed that there will be such a category as well as the possibility of adding remarks without specifically listing these.

(b) It should be a goal that the categories within each field are mutually exclusive. Where this cannot be achieved directly by designation of the categories, one of the following shall be provided: (i) a classification guide that directs the assignment to a specific category wherever there is a possibility that more than one category may apply, or (ii) the establishment of one or more categories for cases that could be assigned to several of the primary categories. As an example consider a failure that caused a primary service output to be slightly late and completely disabled the generation of a maintenance report. A classification guide may direct that where a primary service and a maintenance service are affected the category applicable to the primary service be assigned. Alternatively, additional categories may be established either for multiple effects in general, or for more specific effects, such as: multiple effects, primary no output; multiple effects, primary delayed, etc..

(c) Separate classification files for faults on the one hand and failures on the other are desirable because this facilitates capture of faults not associated with failures, particularly those found in inspections and reviews. Also, the fault file will primarily support the assessment of preventive measures, whereas the failure file will primarily support the assessment of protection and circumvention measures.

(d) To compare the reliability experience from several sites an environment classification file is used to capture the characteristics of the individual sites, such as computer types and configurations, program language, and size of programs.

(e) The number of categories within a given field should be kept to a minimum. Where more than five categories are necessary, consider a two level scheme where categories at the top level are more finely divided at the lower level. This rule can be modified where the number of categories are dictated by external conditions. Typical examples of this are designations of software or hardware components and of the program stage (phase) at which a fault is identified. It is also desirable (but not always feasible) to keep the scope of the categories as uniform as possible so that the a priori expectation is that an equal number of reports will be found in each category.

Fault Classification File

The fault classification is intended to characterize defects that are encountered either by inspection, analytical activities, or execution (operation). The primary purpose of fault classification as described here is to aid in the minimization of future faults. This requires (a) allocation of resources to the areas in which faults reside and the activities associated with detecting them, and (b) technical assessment of causes and corrective measures. The minimum classification topics, covering only the (a) requirements, are:

1. Origin of the fault -- the top level categories are hardware, software, skinware, and environment. Lower level

categories for hardware and software are the component in which the fault was found; for skinware it is the job classification of the responsible person and/or the shift; for the environment it is services (electricity, heating or cooling), fire, natural catastrophes, and hostile acts.

2. Activity responsible for detection -- categories are inspection, review, test, operation, and alarm (for skinware and environmental faults).

3. Identification stage -- program phase at time of detection

Additional classification topics for technical assessment of problems, and particularly for the evaluation of preventive measures, are listed below. The reason for separate listing is that the data for these fields are usually more difficult to get. The order is from the most commonly available to the least available data. In a given application the lower topics can be deleted and there will still remain a usable framework for the assessment of preventive measures.

4. Cause -- a multi-level classification with the top level identical to the origin of the fault. Categories for hardware are: specification or design, manufacture, inspection or test, and installation. Categories for software are: Requirements or specification, design, coding and test. There may be lower levels, particularly for the design category, such as algorithms, structure, data, and interfaces.

5. Isolation -- to identify the activity following detection that is required to proceed with corrective action. Categories are analysis, test, simulation, and special instrumentation.

6. Corrective action -- the top level categories are identical to those of item 1. For software and hardware further subdivisions are desirable for immediate and definitive actions. Immediate actions are: restart (without any other modification), restart with restrictions on operation, switch to an alternate, and replacement of failed component. Definitive actions are: redesign, permanent change in operating procedures, and special testing to detect the responsible condition.

7. Root cause analysis (where available) -- this classification topic can shed light on the project phase during which the fault was introduced. Categories are: requirements, development process, interfaces, and reviews and test. Lower level classification may be warranted in some cases.

Failure Classification

The failure classification is intended to characterize an event and its consequences. The purpose of the classification is to support the assessment of protection and circumvention measures that will minimize the undesirable consequences if the same fault is encountered again. As in the case of the fault classification, it is divided into two parts: (a) the minimum set that is based on directly observable data, and (b) a desirable set that incorporates data that may require further investigations.

The failure data must be obtained for all failures, including those that have no operational effects, such as switchover to a redundant unit, or masked failures. The use of data on failures that have no operational effect is discussed in the application section.

The recommendation for the minimum set is:

1. Failure mode -- the event observed at the computer system level (not at the controlled plant level). Typical categories are: crash, incorrect response, late response, no response, no system effect. This is considered the key entry

because the computer system response is available from test, and most of the target data are presumed to come from test. During test the effect on the controlled system cannot be directly observed, and this is therefore placed in part (b) of this file.

2. Date and time of failure -- required for several types of analyses: (i) sequential ordering of events, (ii) placement of event relative to development milestones or operational changes, (iii) investigation of time of day or workload effects, (iv) reliability growth modeling.

2a. Operation in progress -- the activity during which the failure occurred. Typical categories are: test (may be divided into categories), routine operation, heavy workload, other non-routine operation, maintenance.

2b. Trigger -- record specific events that contributed to the failure. Suggested categories are power transients, other exceptional environment, failure of associated components, unusual operator inputs.

3. Duration (time to restore service) -- important as one indicator of severity of consequences of the failure; this may be a numerical entry or categories such as less than one minute, one to fifteen minutes, and over 15 minutes.

4. Extent of the failure -- expressed in terms of facilities and services affected. Categories are: single channel, multiple channels (less than all), all channels. There can be subdivision for partial failures of channels.

The recommendations for the additional desirable set of data includes:

5. Criticality -- expressed in terms of effect on the controlled plant. In operational environments this can be directly observed and thus become part of the minimum data set. In a test environment analysis and judgment are required to propagate the effects observed at the computer level to the controlled system..

6. Affected component -- identification of the component most directly affected by the failure. This can make use of the multi-level component identification introduced in the fault classification.

7. Detection mechanism -- means by which the first indication of failure was obtained. Top level categories include: alarm or trouble indication, operation of fault tolerance or protective provisions, gross anomaly or cessation of service (without prior alarm), periodic test.

8. Recovery mechanism -- means by which the error was eliminated. Categories include: fault masking, automatic switchover to another component, automatic roll-back or restart, operator intervention, shut-down of service.

9. Restoration mechanism -- required only where recovery does not lead to restoration of service. Categories include: replacement of defective component, re-load or re-initialization of software, and none (where recovery leads to restoration).

10. Prevention and circumvention recommendations -- this is equivalent to the root cause analysis in the fault classification. It is intended to capture the suggestions of personnel who observed a failure on the best means of preventing the effects at the computer level or circumventing the effects at the service or plant level, assuming that the underlying fault or faults similar to it are still present. Categories include: improved monitoring of the computer

operation, adding redundancy within the affected system, prevention (or improved monitoring) of the initiating events, circumvention measures at the plant level such as restrictions on power output under certain conditions.

Environment Classification

The classification of the environment in which the data for the fault and failure files were obtained is essential for comparisons among data obtained at different sites, or when data from different sites are to be combined. The environment file permits accounting for differences in maturity among installations, differences in computer equipment, programming language, plant characteristics, and peculiarities of data collection at a given site. Each file for a given site will result in a record in the environment file.

The major fields of the environment file are:

1. Site identification -- a coded, numeric designation of sites is desirable to assure protection of proprietary data and to avoid ambiguities in free text descriptions.

2. Service function -- typical top level classifications include: air transport, surface transport, space and missile applications, medical services, process industries, manufacturing, and energy.

3. Digital system identification -- coded designation for the reasons described under 1.

4. Number of channels -- number of hardware replications, each one of which is capable of accomplishing the service function

5. Internal redundancy provisions -- within a given channel list number of replications for sensors, sensor communications, data converters external to the computer, computers, output adapters.

6. Computer language (may be subdivided into primary and additional languages) -- typical categories are: assembly, first generation HOLs, structured languages, object oriented languages, and special purpose languages.

6a. Size of developed source code: less than 10k lines, over 10k and up to 30k, over 30k and up to 100k, over 100k.

6b. Fraction of non-developed code -- may be commercial or reused code. Estimate as a fraction of the size of the developed code in the following categories: none, less than 0.05, 0.05 to less than 0.15, 0.15 to less than 0.5, and over 0.5.

7. Development methodology -- typical categories are based on tool usage: compilers and related tools only, static analyzers and related tools, multiple tool usage without dynamic analyzer, multiple tool usage including dynamic analyzer, clean room.

8. Test methodology -- typical categories are: functional test, functional test with complete requirements coverage, functional and structural test with branch coverage of at least 0.95, functional and structural test with path coverage.

9. Independence of V&V -- is represented by the organization from which the V&V team is recruited. Typical categories include: the development organization, another development organization at the same plant, an independent quality assurance organization at the same plant, an outside organization.

10. Maturity of design (applicable to pre-operational systems only) -- expressed in years since start of coding. For system modifications use a weighted average, based on lines of original code and lines of added code.

11. Maturity of system (applicable to operational systems only) -- captured in two fields: number of years since completion of first acceptance test, and total number of installation-years. Both are counted to the start of the failure data collection to which this environment record applies.

12. Maturity of installation (applicable to operational systems only) -- expressed in years since start of operation.

Applications

This section suggests a methodology for applying the classification described above for gaining insight into the failure process at a given site, and for investigating global improvement to reduce the incidence of faults and to improve the protection against failures when they occur. These examples represent only a small sample of possible investigations that will be supported by the classifications described in this report.

Site Specific Applications

Site specific applications are described to illustrate the benefits of a data standard to the management of an individual project, and thus to motivate participation in the broader effort at data standardization and exchange. The association of faults and failures with specific elements of the system is essential for allocating resources for both prevention and mitigation. With respect to faults this is accomplished directly from field 1 (origin of the fault) of the fault classification, and with respect to failures it is accomplished directly from field 6 (affected component) of the failure classification.

Once resources are available, they can be utilized for advancing the time of detection (earlier detection reduces the probability that the fault will lead to severe consequences). For this purpose fields 2 and 3 (activity responsible for detection and identification stage) of the fault file are examined, first with respect to detection of the specific fault, and then with respect to all faults in the component (possibly also in similar components). Detection activities that lead to early detection should be selected in the future.

If the fault resulted in a failure, fields 5 (criticality) and 6 (detection mechanism) of the failure classification are examined. Detection mechanisms that reduce the criticality of failures should be selected in the future.

The probability of a system failure can be computed from the data on non-system failures as shown in the following example.

Assume a two-out-of-four channel safety system; this configuration can sustain two failures and still remain operative but, because a subsequent failure will bring it to an uncertain state, the operating procedures require that the safety system must be declared non-operative after the second failure. For the time being we are not concerned with the effect of this non-operative safety system on the plant.

Over the past twelve months there have been two channel (non-system) failures, one of which took one-half hour to repair and one took four hours to repair. The specific question to be answered: what is the probability of a second failure occurring while the first one is being repaired (plant operating procedures require that in case of a second failure the safety system will be declared non-operative). This probability of failure during the maintenance interval, P_m, can be computed from

$$P_m = \lambda \times \tau \qquad (1)$$

where λ is the failure rate and τ is the repair time (expressed in the same units as the failure rate). In our example the probability of a channel failure is 1/4320 hours $= 231 \times 10^{-6}$ per hour. The average time to repair is 2.25 hours and thus the expectation for a second failure while the first is being repaired is $2.25 \times 231 \times 10^{-6} = 521 \times 10^{-6}$. The first failure is indicated to occur twice a year, and therefore the probability of the system declared to be non-operative is 1.04×10^{-3} per year.

Because of the large variation in repair times it is desirable to augment this expected value calculation by one based on the longest time to repair (which represents a 50% probability in this small sample). In that case the probability of a second failure while the first one is being repaired is $4 \times 231 \times 10^{-6} = 924 \times 10^{-6}$, and the probability of the system being non-operative is 1.85×10^{-3} per year.

Global Applications

This topic is concerned with two broad application areas: drawing inferences from differences in fault or failure data based on environment characteristics, and merging data from different sites to form an aggregate database.

An example of the first type, drawing inferences based on environment characteristics, is at first glance very simple as shown in the following example. Two projects were started at about the same time, and after three years both have completed acceptance test. Project X has a fault density of 5 per 1000 non-comment source lines and project Y has a fault density of 2.5 per 1000 non-comment source lines. Project X was written in language A and project Y in language B. The hypothesis is formulated that language B leads to lower fault density than language A. Should this hypothesis be accepted?

A conventional approach to answering this question is to utilize statistical methods of testing hypotheses. But an underlying assumption in all of these is that the "treatments" being compared (in our case the two computer languages) are the only differences between the alternatives. This assumption is practically never valid for software development environments. Therefore the statistical procedures discussed below should be considered a clue rather than evidence of a possible cause for the observed differences. Detailed comparisons of various elements of the software product and of the processes utilized must then be used to confirm or refute the statistical results.

Table 1. Hypothetical Comparison

Component or Computed Quantity	Project X	Project Y
	Fault Density, k lines	
1	6	0
2	7	8
3	4	0
4	3	9
5	5	2
6	4	1
7	6	0
8		0
n	7	8
$\sum\xi$	35	20
m	5	2.5
R	4	9
$\sum\xi^2$	187	150
σ	1.3	3.8

Because of the restricted role assigned to the statistical investigation it is rarely worthwhile to compute statistical confidence limits. The rules of thumb presented here are simpler (some might say cruder) and are drawn from the experience of the authors but they are based on the same fundamental principle as conventional statistical analysis: consider the hypothesis proven only if the difference is large compared to the "noise" in the data. For the small number of observations that are typical of this environment the noise is measured in the simplest approach by the range[1], or in a more sophisticated approach by the internal standard deviation, of the measured variable within each population, in this case the fault density in project X and project Y. In our example the project X software consisted of seven major components and the project Y software of eight. The individual fault densities are shown in the following table,

[1] For larger populations the difference between quartiles should be substituted for the range.

together with key computed results from each data set.

Among the computed quantities (bottom of the table), n is the number of major components in each project, $\sum\xi$ is the sum of the component fault densities, m is the mean of the component fault densities[2] computed from $m = \sum\xi/n$, and R is the range (difference between the highest and lowest fault density for each project. For a significant statistical difference to exist the two treatments should differ by at least one-half of the average range. In this case the average range is 6.5 and the difference in fault density is only 3. Thus this test does not indicate a basis for accepting the hypothesis.

The variance is computed from $\sigma^2 = (\sum\xi^2/n) - m^2$ and the standard deviation, σ, represents the square root of the variance. As a rule of thumb a statistically significant difference exists only if the two observations (here the mean of the fault densities) differ by at least the larger of the two standard deviations. This criterion is also not met here, and thus there is no basis for accepting the hypothesis.

Where the failure rates of two projects are compared the noise content can be measured in terms of the intervals between failures. Assume that failures are observed at intervals of 100, 200, 20, 80, and 200 hours (five failures during 600 hours of operation). The mean time between failures is 120 hours, and the corresponding failure rate is 1/120 = 0.0083 per hour. The half-range is 90 hours, and at least that difference should by our criteria be observed to accept that another project has significantly superior or inferior product characteristics. The standard deviation can also be computed by the methodology outlined above and the same criteria can then be applied.

Technically motivated data merging may aim at increasing the number of observations by combining fault or failure files from several projects to facilitate statistical comparisons. Suppose that instead of the single projects with languages A and B that were discussed above there were four that used each of these languages. The resulting larger data sets normally reduce the noise and thus increase the likelihood of positive results when hypotheses are tested. Merging for this objective requires considerable care in the evaluation of environment factors. This can be accomplished by evaluating at least those items in the environment file which have been identified as contributing to differences in fault density, failure rate, or related characteristics: function and size of the software, characteristics of the hardware, maturity of hardware and software, and the year the project was initiated.

In addition, statistical criteria can be used to determine whether two data sets come from a homogenous population. In principle this is just the reverse of the process described to show that populations are distinct: it must be shown that the difference in the means is within the noise boundaries. The limits that will be set for acceptance depend on the purpose for which the combined data will be used. If a voluntary recommendation for future use is to be formulated data sets can be combined with more generous limits than when mandatory acceptance criteria are to be generated.

[2] This will usually be different from the mean software fault density which is obtained by dividing the total number of faults by the total number of lines of code.

References

[FAA88] Federal Aviation Administration, Advisory Circular 25.1309-1A, "System Design and Analysis", June 1988

[FAA91] Federal Aviation Administration, Product Specification for the Voice Switching and Control System, FAA-E-2731F, October 1991

[HECH95] H. Hecht, M. Hecht, G. Dinsmore, S. Hecht, D. Tang, "Verification and Validation Guidelines for High Integrity Systems", NUREG/CR-6293, March 1995

[MUSA87] John Musa, Iannino,A. and Okumoto, K., *Software Reliability Measurement Prediction, Application*, McGraw-Hill Co. 1987